G-MACT at SemEval-2025 Task 8: Exploring Planning and Tool Use in Question Answering over Tabular Data

Wei Zhou[1,2] **Mohsen Mesgar**[1] **Annemarie Friedrich**[2] **Heike Adel**[3]
[1]Bosch Center for Artificial Intelligence, Renningen, Germany
[2]University of Augsburg, Germany [3]Hochschule der Medien, Stuttgart, Germany
{wei.zhou|mohsen.mesgar}@de.bosch.com
annemarie.friedrich@uni-a.de adel-vu@hdm-stuttgart.de

Abstract

This paper describes our system submitted to SemEval-2025 Task 8 "Question Answering over Tabular Data." The shared task focuses on tackling real-life table question answering (TQA) involving extremely large tables with the additional challenges of interpreting complex questions. To address these issues, we leverage a framework of Multi-Agent Collaboration with Tool use (MACT), a method that combines planning and tool use. The planning module breaks down a complex question by designing a step-by-step plan. This plan is translated into Python code by a coding model, and a Python interpreter executes the code to generate an answer. Our system demonstrates competitive performance in the shared task and is ranked 5th out of 38 in the open-source model category. We provide a detailed analysis of our model, evaluating the effectiveness and the efficiency of each component, and identify common error patterns. Our paper offers essential insights and recommendations for future advancements in developing TQA systems.

1 Introduction

Table question answering (TQA) focuses on addressing questions related to tables. It has been widely studied across domains (Zhu et al., 2021; Lu et al., 2023; Katsis et al., 2022) and languages (Zheng et al., 2023; Pal et al., 2024; Jun et al., 2022). Current TQA datasets predominantly feature tables from Wikipedia (Pasupat and Liang, 2015; Zhang et al., 2023), resulting in an oversimplified task setup characterized by small, clean tables with limited diversity in data types. To promote studies in real-life TQA, Osés-Grijalba et al. (2025) propose DataBench, an English TQA dataset consisting of 65 tables and around 1300 manually created questions spanning various domains. Based on this dataset, the SemEval-2025 Task 8 "Question Answering over Tabular Data" encourages TQA modeling in a more realistic and challenging setup. It consists of two subtasks: ALL, where original long tables are used and LITE, in which only the first 20 rows of a table are used.

There are two main challenges in the task: (1) the **large table size** makes direct inference using large language models (LLMs) difficult due to their limited input lengths and problems of being *lost in the middle* (Liu et al., 2024). (2) The **complexity of questions** requires multiple steps to be solved. For instance, to answer the question in Figure 1, a system should first filter for employees who are working in sales and then calculate the average of working years among those employees.

To address these challenges, we propose G-MACT, a method combining Global planning with Multi-Agent Collaboration with Tool use (Zhou et al., 2025). G-MACT comprises two modules: a global and an iterative planning module. The global planning module takes in the first several rows of a table alongside a question, and generates a step-by-step plan. By doing this, we break down a complex question into easier sub-steps. A coding agent translates this plan into pandas-based Python code,[1] which is executed using a Python interpreter to derive an answer.

Despite being efficient, the global planning module may encounter difficulties in formulating an optimal plan when it relies solely on a partial table and a question. To enhance the robustness of plan generation by conditioning it also on past observations, we apply the iterative planning framework MACT whenever the global planning module does not yield an answer. The framework generates one step of a plan in each iteration, considering past steps and observations. We ensemble results from four models for our final submission. Our method ranks 5th among 38 open-weight systems in the challenging ALL setup.

[1]https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025), pages 726–742

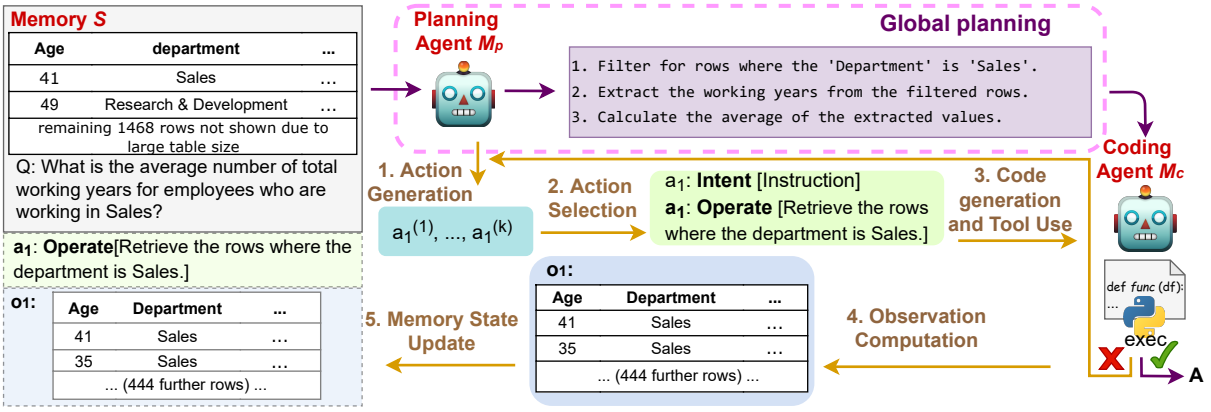

Figure 1: G-MACT illustration. It leverages both global planning (shown in purple) and iterative planning (shown in orange).Iterative planning is applied if Python fails to compile and no answer is obtained.

We thoroughly analyze G-MACT from three perspectives: effectiveness of the global and iterative planning modules, efficiency, as well as error patterns. We find that most instances in DataBench can be solved by global planning and tool use alone. However, the iterative planning module proves essential for instances where global planning fails to yield an answer. Additionally, our findings underscore the significance of selecting an optimal ensemble method. We observe for around 91% of the cases, at least one model predicted the correct answer, while the ensemble method used in our approach only achieves around 86% of exact match (EM) accuracy. This discrepancy highlights a clear need for exploring more sophisticated ensembling techniques. Lastly, through an error analysis, we find that developing a good planning agent that can understand table semantics and has access to factual and domain-specific knowledge is of vital importance for further enhancing a model's performance. We make our system publicly available.[2]

2 Related Work

The main methodologies used by G-MACT are planning and tool use. Both have been employed in TQA to facilitate more fine-grained problem solving, thus improving model performance (Zhao et al., 2024; Wu and Feng, 2024; Zhou et al., 2025; Wang et al., 2024). Current research either utilizes global planning (Zhao et al., 2024) or iterative planning (Wang et al., 2024; Zhou et al., 2025). Global planning involves generating a plan consisting of multiple steps in a single iteration, conditioned solely on a question and a table. Iterative planning conditions the generation of the next step of a plan on previous observations and steps. While global planning tends to be more efficient, iterative planning offers more fine-grained plan generation. In G-MACT, we integrate both planning methods.

3 System Overview

Given a table T and a question Q, a TQA system aims to address Q and return an answer A. Our proposed system G-MACT combines global planning with the iterative approach of multi-agent collaboration with tool use (Zhou et al., 2025) in a pipeline manner as illustrated in Figure 1. In this section, we introduce the global planning module, the iterative planning module, and the ensemble method used to create the submission results.

3.1 Global Planning

Our global planning module comprises a planning agent M_p and a coding agent M_c with a Python interpreter. As shown in Figure 1, M_p takes in the first two rows of a table and a question, then generates a step-by-step plan. This can be represented as: $P \sim M_p(P|Q, T', \phi_p, \tau_p)$, where T' is a subpart of a table. ϕ_p and τ_p are the prompt (provided in A.1) and the temperature of the LLM used for M_p, respectively. We pass only the first two rows of a table because (1) LLMs have been shown to struggle with long tables (Zhou et al., 2024) and (2) plan generation requires mainly information about the table columns and data types, which can be derived from the columns and first two rows of the table. We utilize in-context learning to prompt M_p to generate a step-by-step plan, providing instructions to solve a question, without intermediate results. An example is shown in the purple box of Figure 1. Given a plan P, a cod-

[2] https://github.com/boschresearch/MACT

ing agent M_c generates Python code using pandas: $c_i \sim M_c(c_i|P, T', Q, \phi_c, \tau_c)$. We sample k times from M_c to increase the robustness of the system against generated syntax errors, resulting in a set of code snippets $C = \{c_i^n\}_{n \leq k}$. A Python interpreter is run on each c_i, creating a set of executed solutions $\hat{A} = \{\hat{a}_i^n\}_{n \leq k}$. The final answer A is the most frequent answer in $\hat{A}$.

3.2 Iterative Planning

The global planning module can be very efficient since each question requires only one LLM call to generate a plan. However, it is still possible that no prediction is given by the module, namely if no code generated from M_c is successfully executed. To mitigate the impact of failed execution, we resort to an iterative planning module if no answer is given by the global planning module. The design of the module is based on MACT (Zhou et al., 2025). The framework takes in a TQA problem, i.e., a full table and a question, and returns a prediction. This is achieved by breaking down a complex problem into fine-grained steps and addressing each step with two agents (a planning M_p, a coding agent M_c) and a toolset. Zhou et al. (2025) define each step as an intent and an instruction. An intent encodes the purpose of a step and the instruction provides detailed specifications of the intent. For addressing each step, M_p and M_c perform two layers of collaboration: (1) M_c takes in instructions given by M_p for code generation. (2) The final step solution is determined as the most frequent result from $\{\hat{o}_i^n\}_{n \leq k} \cup \hat{C}$, where $\{\hat{o}_i^n\}_{n \leq k}$ are step solutions generated by M_p and $\hat{C}$ is based on executing Python code generated by M_c. Note that in MACT, the generation of a next step depends on previous steps, thus being iterative and more computationally expensive. To boost efficiency, MACT features an efficiency optimization module, where simple questions are directly answered by M_p without going into the iterative loop. Zhou et al. (2025) approximate question complexity by the confidence of M_p in directly solving a TQA problem: a confident model will output more agreed predictions and this suggests the problem is less complex.

We adapt MACT for the shared task as follows: (1) MACT requires a whole table as input. As this is not possible with large tables, we only pass the first two rows for each step and code generation. Accordingly, we adapt the prompts for M_p and M_c used in the iterative planning module. These are presented in Appendix A.1. (2) We remove the efficiency module in MACT. This module requires a full table to generate the final answer. In our case, since only the first two table rows are passed to the system, the answer predicted by the efficiency module cannot be trusted. (3) We remove the layer of collaboration where step solutions are determined by both M_p and M_c and select the most frequent observation from $\hat{C}$, as step solutions generated by M_p might not be correct given only two table rows. (4) We merge the intent *Retrieve* and *Calculate* into *Operate* to increase efficiency since both use Python and M_c in this case. We remove intents *Search* and *Read* where Wikipedia search and LLM extraction over texts happen, as DataBench does not have additional text input and does not feature open-domain TQA. If no answer is obtained from the iterative planning module, we return *none* as the final answer.

3.3 Ensemble Method

For our submissions, we ensemble results from four different M_p and one M_c. Since each M_p and M_c combination yields an answer A for a TQA instance, we have four predictions for an instance. We design our ensemble method as a combination of self-consistency (sc)(Wang et al., 2023) and LLM-as-judge (Yao et al., 2023): If more than 60% of the predicted answers are the same, we use the most agreed answer as the final answer (sc). If less than 60% of the predicted answers are the same, we prompt an LLM to select the most reasonable plan. Prompts can be found in Appendix A.1. The corresponding answer obtained by executing the selected plan is chosen as the final answer.

4 Experimental Setup

We present details on our experimental setup.

Data and Evaluation. DataBench (Osés-Grijalba et al., 2025) includes 65 English tables from diverse domains, with an average of over 3,200 rows and 1,600 columns. Each table is accompanied by more than 20 manually created questions, resulting in approximately 1,300 questions in total. The questions vary in terms of answer types including boolean, category, number, list[category], and list[number]. We use only the test set of DataBench, which contains 15 tables and 522 questions. Detailed statistics are presented in Appendix A.2. We use exact match (EM) as evaluation metric, which counts the percentages of

Models	ALL						LITE					
	Avg	Bool	Ctg	Num	[ctg]	[num]	Avg	Bool	Ctg	Num	[ctg]	[num]
	522	129	74	156	72	91	522	129	74	156	72	91
Qwen-2 (72B)	80.1	89.1	75.7	81.4	69.4	76.9	78.5	89.1	77.0	78.8	66.7	73.6
Deepseek (14B)	81.6	88.4	81.1	85.3	69.4	75.8	81.8	86.8	79.7	84.6	72.2	79.1
Mistral (13B)	75.5	89.1	70.3	77.6	56.9	71.4	78.0	87.6	73.0	81.4	69.4	69.2
LlaMA-3 (8B)	70.1	89.1	66.2	67.3	59.7	59.3	74.7	90.7	74.3	76.3	61.1	60.4
Ensemble	**86.0**	91.5	82.4	78.2	73.6	80.2	**84.5**	89.1	86.5	85.9	73.6	82.4

Table 1: Exact Match of G-MACT using different models as the planning agent in terms of answer category. Ctg and Num stand for category and number, respectively. [x] refers to a list with items of type x. We report the instance number of each answer type at the top part of the table.

predicted and reference answers that match exactly. We use the official evaluation scripts provided by Osés-Grijalba et al. (2025).

Models and Parameters. We use four different LLMs as planning agents: Qwen-2-int4 (72B) (Yang et al., 2024a), Mistral Nemo (13B),[3] Deepseek-R1-distill-Qwen (14B) (DeepSeek-AI et al., 2025), and LlaMA 3 (8B) (Dubey et al., 2024). As coding agent, we use Qwen-2.5-coder (32B) (Yang et al., 2024b). This results in four possible pairs of planning and coding agents. We set the sampling number k to 5. The temperature is set to 0.6. To speed up inference, we use vllm[4] to run M_p. M_c is deployed with SGLang.[5] We use Deepseek-R1-distll-Qwen (32B) (Yang et al., 2024b) as the judge to choose the best plan in the ensemble method.

Baselines. We compare G-MACT with top four systems in the open-weight model category in SemEval-2024 Task 8, with a focus on the more challenging ALL setup. In addition, we compare our method with the baseline reported by Osés-Grijalba et al. (2025), where stable-code[6] is used to generate code and a Python interpreter executes the code to obtain an answer.

5 Results

Figure 2 shows the performance of G-MACT compared with top four systems and the baseline reported in Osés-Grijalba et al. (2025) for the more challenging ALL setup. Our ensemble model outperforms the baseline method by a large margin. However, there is still a gap between our method and the best-performing system in the shared task.

[3] https://mistral.ai/news/mistral-nemo/

[4] https://github.com/vllm-project/vllm

[5] https://github.com/sgl-project/sglang

[6] https://huggingface.co/TheBloke/stable-code-3b-GGUF

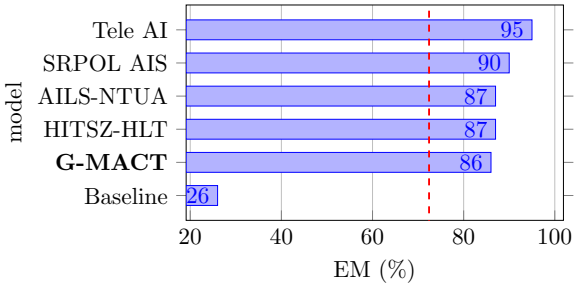

Figure 2: Comparing G-MACT with top four systems in the open-weight model category in the ALL setup. We also report baselines (ranked 33) provided by Osés-Grijalba et al. (2025). The red dotted line (72.4) indicates the median performance.

Table 1 shows results using different planning models in ALL and LITE settings (see columns title **Avg**). We find: (1) Ensemble results from different models improve overall performances. (2) Using Deepseek-Distill-Qwen (14B) as planning agent leads to the best results among individual planning-coding agent pairs. This might be attributed to the model's recency, its training mechanism, and its pretraining data (DeepSeek-AI et al., 2025). When looking at break-down results in terms of answer categories, we find that for almost all models and settings, questions that require a list of categorical values as answers pose the biggest challenges. This is followed by questions that ask for a list of numbers as answers. In contrast, questions with boolean answers are the easiest. Osés-Grijalba et al. (2025) report similar observations. This suggests that multi-value prediction poses unique challenges to current TQA systems.

6 Analysis and Discussion

We analyze G-MACT in terms of planning, ensembling, efficiency, and errors, and summarize key insights for future studies.

	ALL		LITE	
	EM	Global%	EM	Global%
Global	76.4	100	77.8	100
Iterative	64.2	0	63.4	0
Both	**81.6**	91.0	**81.8**	92.0

Table 2: Exact Match (EM) of each single module and combined, as well as the percentages of instances addressed by using the global planning module with Deepseek-Distill-Qwen (14B) as the planning agent.

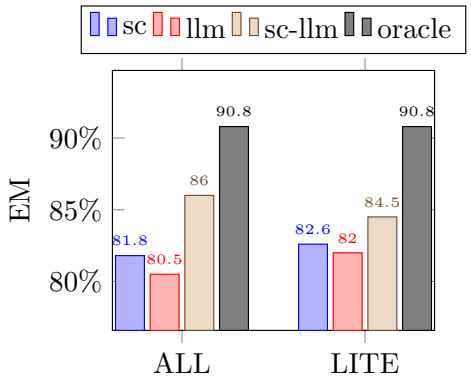

Figure 3: Exact Match (EM) of using different ensemble methods. sc=self-consistency. llm=LLM-as-a-judge. oracle=an ensemble method that always selects correct answers as final predictions if there are any.

Effectiveness of Global/Iterative Planning. To assess the effectiveness of the global/iterative planning modules, we experiment with using each module independently. We also calculate the proportion of instances that are successfully addressed by only applying the global planning module. These are shown in Table 2. Results are obtained using Deepseek-Distill-Qwen (14B) model, as it demonstrates the best overall performances in both ALL and LITE settings among investigated planning models. We find that most instances can be addressed using only the global planning module. However, incorporating the iterative module significantly enhances performance by 9.6% and 6.7% in the ALL and LITE settings, respectively. This proves the effectiveness of combining both modules. Despite these gains, we observe that the iterative planning module alone results in lower performances compared to the global planning approach.

Ensemble Methods. We report EM achieved by our ensemble method combining sc and LLM-as-judge, as well as each individual method in Figure 3. We present an EM upper bound of ensembling the four models, which is calculated as the percentage of correct answers in any of the four models' predictions. We find that combining both sc and LLM-as-judge leads to better results than using them alone. However, there is still a gap between our ensemble method and the potential best ensemble approach (oracle), indicating that a better confidence estimation for the answers provided by each individual component of the ensemble could lead to considerable improvements.

Efficiency Analysis. The global planning module requires six LLM calls (one for planning and five for code generation) for each instance. As shown in Table 2, most instances can be addressed by applying global planning alone. For those requiring additional iterative planning, we observe that most instances can be addressed within two iterations. This is shown in Appendix A.3). This means most instances only require 15 LLM calls.[7]

Error Analysis. We manually analyze and summarize error types among instances whose predicted answers are wrong by all four models in both settings. This results in 96 instances in total. Around 50% of errors are caused by **wrong plan generation**, which includes incorrect question interpretation (e.g., selecting wrong features for computing), failure to understand table semantics (e.g., the column *Tier 1* is the parent node of the column *Tier 2*), and incorporating factual or domain-specific knowledge (e.g., in basketball, *OREB* stands for offensive rebounds, where the ball is recovered by the offensive side and does not change possession). Another 20% of the errors can be attributed to **incorrect semantic matching between questions and tables**, e.g., the entities mentioned in the question might not exactly match the entities in the tables. Using only the first two rows of a table can exacerbate the problem of matching semantically similar entities in a question and a table, e.g., "books about computer science" in the question and "Computer Science & Engineering" in the table. The challenge can be addressed by utilizing more flexible row selection methods. For instance, instead of passing the first two rows where no category name of computer science is shown, one can use semantic matching between a question and rows to select the most relevant rows that contain "Computer Science & Engineering".

[7] 5*2=10 LLM calls for iterative planning. For code generation, only 5 LLM calls are needed since the last iteration does not require tool calling and only returns a final answer.

By doing this, the coding model is more likely to generate correct filtering conditions. Similar ideas have been explored in Chen et al. (2024). Due to limited time, we leave this for future exploration. Interestingly, fewer errors are caused by **code generation and execution** (10%) and most of them can be solved by data cleaning beforehand, e.g., aligning the categories encoded in a categorical header with the values in the column. This might be because code about common table operations, e.g., filtering, is easy to generate given clear textual instructions. Lastly, around 20% of the errors come from **question ambiguity**. We provide error examples for each category in Appendix A.4.

Takeaway Messages. Combining global and iterative planning in TQA is effective and worth exploring. Designing a good planner that understands table semantics and has access to factual and domain-specific knowledge is crucial. Ensembling different models increases overall performance. Moreover, how to select the best model/plans is decisive for improving ensemble results.

7 Conclusion

In this paper, we introduce G-MACT, a pipeline framework combining planning and tool use, developed for the SemEval-2025 Task 8. Our method ranks 5th among all approaches using open-weight models, with no training involved. We carefully analyze our system in terms of the effectiveness of each module, efficiency, and error patterns. We provide key insights for future work to address real-life table question answering.

Acknowledgments

This work was partially supported by the EU Project SMARTY (GA 101140087).

References

Si-An Chen, Lesly Miculicich, Julian Martin Eisenschlos, Zifeng Wang, Zilong Wang, Yanfei Chen, Yasuhisa Fujii, Hsuan-Tien Lin, Chen-Yu Lee, and Tomas Pfister. 2024. Tablerag: Million-token table understanding with language models.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bing-Li Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dong-Li 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, Jiong 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, M. 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, Shaokang Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wen-Xia 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, Xiangyu Jin, Xi-Cheng 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, Yi Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yu-Jing Zou, Yujia He, Yunfan Xiong, Yu-Wei Luo, Yu mei You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yao Li, Yi Zheng, Yuchen Zhu, Yunxiang Ma, Ying Tang, Yukun Zha, Yuting Yan, Zehui Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhen guo Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zi-An 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.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony S. Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Bap tiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Cantón Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab A. AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriele Synnaeve, Gabrielle

Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guanglong Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Laurens Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Ju-Qing Jia, Kalyan Vasuden Alwala, K. Upasani, Kate Plawiak, Keqian Li, Ken-591 neth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuen ley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Babu Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melissa Hall Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri S. Chatterji, Olivier Duchenne, Onur cCelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasić, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Ro main Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Chandra Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whit ney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yiqian Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zhengxu Yan, Zhengxing Chen, Zoe Papakipos, Aaditya K. Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adi Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Ben Leonhardi, Po-Yao (Bernie) Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Shang-Wen Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzm'an, Frank J. Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory G. Sizov, Guangyi Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Han Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kaixing(Kai) Wu, U KamHou, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, A Lavender, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollár, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma,

Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sung-Bae Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Andrei Poenaru, Vlad T. Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xia Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. 2024. The llama 3 herd of models. *ArXiv*, abs/2407.21783.

Changwook Jun, Jooyoung Choi, Myoseop Sim, Hyun Kim, Hansol Jang, and Kyungkoo Min. 2022. Korean-specific dataset for table question answering. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 6114–6120, Marseille, France. European Language Resources Association.

Yannis Katsis, Saneem Chemmengath, Vishwajeet Kumar, Samarth Bharadwaj, Mustafa Canim, Michael Glass, Alfio Gliozzo, Feifei Pan, Jaydeep Sen, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2022. AIT-QA: Question answering dataset over complex tables in the airline industry. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, pages 305–314, Hybrid: Seattle, Washington + Online. Association for Computational Linguistics.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Xinyuan Lu, Liangming Pan, Qian Liu, Preslav Nakov, and Min-Yen Kan. 2023. SCITAB: A challenging benchmark for compositional reasoning and claim verification on scientific tables. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7787–7813, Singapore. Association for Computational Linguistics.

Jorge Osés-Grijalba, Luis 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)*, Vienna, Austria. Association for Computational Linguistics.

Vaishali Pal, Evangelos Kanoulas, Andrew Yates, and Maarten de Rijke. 2024. Table question answering for low-resourced Indic languages. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 75–92, Miami, Florida, USA. Association for Computational Linguistics.

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, Beijing, China. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *ICLR*.

Zirui Wu and Yansong Feng. 2024. ProTrix: Building models for planning and reasoning over tables with sentence context. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4378–4406, Miami, Florida, USA. 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, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Ke-Yang Chen, Kexin Yang, Mei Li, Min Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yunyang Wan, Yunfei Chu, Zeyu Cui, Zhenru Zhang, and Zhi-Wei Fan. 2024a. Qwen2 technical report. *ArXiv*, abs/2407.10671.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan,

and Zekun Wang. 2024b. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate problem solving with large language models.

Zhehao Zhang, Xitao Li, Yan Gao, and Jian-Guang Lou. 2023. CRT-QA: A dataset of complex reasoning question answering over tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2131–2153, Singapore. Association for Computational Linguistics.

Yilun Zhao, Lyuhao Chen, Arman Cohan, and Chen Zhao. 2024. TaPERA: Enhancing faithfulness and interpretability in long-form table QA by content planning and execution-based reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12824–12840, Bangkok, Thailand. Association for Computational Linguistics.

Mingyu Zheng, Yang Hao, Wenbin Jiang, Zheng Lin, Yajuan Lyu, QiaoQiao She, and Weiping Wang. 2023. IM-TQA: A Chinese table question answering dataset with implicit and multi-type table structures. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5074–5094, Toronto, Canada. Association for Computational Linguistics.

Wei Zhou, Mohsen Mesgar, Heike Adel, and Annemarie Friedrich. 2024. FREB-TQA: A fine-grained robustness evaluation benchmark for table question answering. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2479–2497, Mexico City, Mexico. Association for Computational Linguistics.

Wei Zhou, Mohsen Mesgar, Annemarie Friedrich, and Heike Adel. 2025. Efficient multi-agent collaboration with tool use for online planning in complex table question answering.

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. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3277–3287, Online. Association for Computational Linguistics.

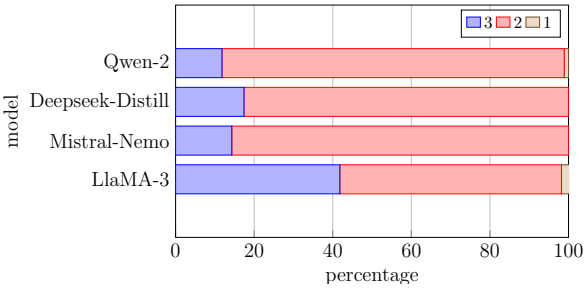

Figure 4: Average iteration distribution over settings.

A Appendix

A.1 Prompts

We show prompts used for a planning agent in the global and iterative planning modules in Figure 5 and Figure 6, respectively. We show prompts used for a coding agent in the global and iterative planning modules in Figure 7 and Figure 8, respectively. Lastly, we show the prompt used to select the best plan in our ensemble method in Figure 9.

A.2 Data Statistics

Table 3 shows statistics about the test set of DataBench.

A.3 Efficiency Analysis

We plot the number of steps required in the iterative planning module to solve a question, averaging over two settings in Figure 4.

A.4 Error Examples

We show four concrete examples of errors made using G-MACT in Figure 10, 11, 12 and 13, with each mapping an aforementioned category.

You are an expert in analyzing table data and generate step-by-step plans to solve any questions related to long tables.
The following table only shows the first three rows of the table due to its large size.
Please generate a step-by-step plan to address the question, following the below requirement:
1. A plan should contain no more than 4 steps.
2. Each step should be in one line.
3. Return only the step-wise plan and nothing else.
4. No repetition of the plan.
Please return only a plan and nothing else.
Following are three examples:
Table:
| rank | personName | age | finalWorth | category | source | country |…
| 1 | Elon Musk | 50.0 | 219000 | Automotive | Tesla, SpaceX | United States |…
| 2 | Jeff Bezos | 58.0 | 171000 | Technology | Amazon | United States ...
…[remaining 2665 rows unshown due to large table size]...
Context: Table caption: Forbes Billionaires 2022.
Question: How many billionaires are there from the 'Technology' category?
Plan: 1. I need to filter the table to get all billionaires from the 'Technology' category.
2: Then I need to count the number of retrieved entries.
3. The answer to the question is the number of retrieved entries in the second step, and I will return this value as the final answer.

Table:
| segmentation_1 | descriptor | complaint_type | created_date |…
| [] | Business not in compliance | NonCompliance with Phased Reopening | 2020-10-02 11:50:27+00:00 |…
| [] | ENTIRE BUILDING | HEAT/HOT WATER | 2021-11-03 06:48:49+00:00 |…
...[remaining 99997 rows unshown due to large table size]...
Context: Table caption: NYC 311 Calls
Question: Mention the 2 most common weekdays for complaints.
Plan: 1. I need to count the frequency of each unique weekday in the column 'weekday_name'.
2. I will create an additional dataframe with two columns to store the results from the first step, with one column being the name of the weekday, and one being the frequency of that weekday.
3. I will sort the dataframe I made in step 2 in descending order.
4. The question asks 2 most common weekdays, this corresponds to the weekday values of the top two rows. I will retrieve the weekday values of the top two rows, store them in a list and return the list as the answer.

Table:
| segmentation_1 | descriptor | complaint_type | created_date |…
| [] | Business not in compliance | NonCompliance with Phased Reopening | 2020-10-02 11:50:27+00:00 |…
| [] | ENTIRE BUILDING | HEAT/HOT WATER | 2021-11-03 06:48:49+00:00 |…
...[remaining 99997 rows unshown due to large table size]...

Context: Table caption: NYC Taxi Trips.
Question: Are there any trips with a total distance greater than 30 miles?
Plan: 1. I need to count the number of entries whose 'trip_distance' is larger than 30.
2. If the value from step 1 is larger than 0, then the answer is 'True', otherwise, it is 'False'.
3. I will create a variable name after 'final_result' to store the boolean answer and return the variable as final answer.

Now generate a plan for to address the following question and table. The plan should contain maximum 4 steps, with each step one line.
Table: {table}
Context: {context}
Question: {question}

Figure 5: A prompt for a planning agent in the global planning module.Caption

Solve a table question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be two types:
(1) Operate[instruction], which carries out operations such as information retrieval or calculations based on the instruction and returns the retrieved or calculated results.
(2) Finish[answer], which returns the answer and finishes the task.
You may take as many steps as necessary.
Here are some examples:
Table:
| rank | personName | age | finalWorth | category | source | country |…
| 1 | Elon Musk | 50.0 | 219000 | Automotive | Tesla, SpaceX | United States |…
| 2 | Jeff Bezos | 58.0 | 171000 | Technology | Amazon | United States |...
…[remaining 2665 rows unshown due to large table size]...
Context: Table caption: Forbes Billionaires 2022.
Question: How many billionaires are there from the 'Technology' category?
Thought 1: I need to count the number of billionaires from the 'Technology' category.
Action 1: Operate[count the number of entries whose category is Technology]
Observation 1: 343
Thought 2: In observation 1, 343 billionaires are from the 'Technology' category, therefore, the answer is 343.
Action 2: Finish[343]

Table:
| segmentation_1 | descriptor | complaint_type | created_date |…
| [] | Business not in compliance | NonCompliance with Phased Reopening | 2020-10-02 11:50:27+00:00 |…
| [] | ENTIRE BUILDING | HEAT/HOT WATER | 2021-11-03 06:48:49+00:00 |…
...[remaining 99997 rows unshown due to large table size]...
Context: Table caption: NYC Taxi Trips.
Question: Are there any trips with a total distance greater than 30 miles?
Thought 1: I need to count the number of entries whose 'trip_distance' is larger than 30.
Action 1: Operate[count the number of entries whose 'trip_distance' is larger than 30.]
Observation 1: 0
Thought 2: In observation 1, there is 0 entry whose trip distance is larger than 30. Therefore, the answer is False.
Action 2: Finish[False]

Table:
| segmentation_1 | descriptor | complaint_type | created_date |…
| [] | Business not in compliance | NonCompliance with Phased Reopening | 2020-10-02 11:50:27+00:00 |…
| [] | ENTIRE BUILDING | HEAT/HOT WATER | 2021-11-03 06:48:49+00:00 |…
...[remaining 99997 rows unshown due to large table size]...
Context: Table caption: NYC 311 Calls
Question: Mention the 2 most common weekdays for complaints.
Thought 1: I need to count the frequency of each unique weekday in the column 'weekday_name'.
Action 1: Operate[count the frequency of each unique weekday in the column 'weekday_name'.]
Observation 1: {'Tuesday': 15847, 'Monday': 15816, 'Wednesday': 15445, 'Thursday': 14978, 'Friday': 14707, 'Saturday': 11781, 'Sunday': 11426}
Thought 2: The question ask for 2 most common weekdays. From observation 1, we find Tuesday and Monday have the largest frequencies and they are weekdays. Therefore, the answer is ["Tuesday", "Monday"]
Action 2: Finish[["Tuesday", "Monday"]]
(END OF EXAMPLES)
Now generating the Thought, Action, Observation for the following instance:
Table:
{table}
Context: {context}
{question}
{scratchpad}"""

Figure 6: A prompt for a planning agent in the iterative planning module.

You are an expert in python code generation.
Write a python function named 'target_function' according to the given plan using pandas dataframe.
The given dataframe shows only two records of the original data due to its large size. The main goal of showing the dataframe is to show the data type associated to each column.
However, you should not operate any code based on the given dataframe, since it does not contain all information about the table.
Below are two examples
Plan: 1. I need to filter the table to get all billionaires from the 'Technology' category.
2: Then I need to count the number of retrieved entries.
3. The answer to the question is the number of retrieved entries in the second step, and I will return this value as the final answer.
Dataframe code for the first two records: import pandas as pd
data={'rank':[1.0, 2.0],'personName':['Elon Musk', 'Jeff Bezos'],'age':[50.0, 58.0],'finalWorth':[219000.0, 171000.0],'category':['Automotive', 'Technology'],'source':['Tesla, SpaceX', 'Amazon'],'country':['United States', 'United States'],…}
df=pd.DataFrame(data)
Code: ```Python
def target_function(dataframe):
  # filter the table for 'Technology' as the category and count the number of the entries
  technology_entries_count = len(dataframe[dataframe['category'] == 'Technology])
  # return the result as final answer
  return technology_entries_count
```

Plan: 1. I need to retrieve the first five values from the 'Gold' columns.
2. To calculate the average number, I will sum the retrieved values and divide the sum by 5.
3. The answer to the question is the result from step 2. I will return that value as the final answer.
Dataframe code for the first two records: import pandas as pd
data={"Rank": ["1", "2"], "Nation": ["United States", "Jamaica"], "Gold": ["5", "4"], "Silver": ["6", "1"], "Bronze": ["5", "1"], "Total": ["16", "6"]}
df=pd.DataFrame(data)
Code: ```Python
def target_function(dataframe):
  # retrieve the top 5 gold medals values from the table
  top_5_medals = dataframe["Gold"].astype(int).tolist()[:5]
  # get the average number of the gold medal
  final_result = sum(top_5_medals) / 5
  # return the result
  return final_result
```
Now generate the python function according to the given plan.
Plan: {instruction}
Dataframe code for the first two records: {table_df}
Code:

Figure 7: A prompt for a coding agent in the global planning module.

According to the instruction, write a function named after 'target_function' in one python code block to perform calculations on a dataframe object. The given dataframe shows only two records of the original data due to its large size. However, you should be able to infer the data type based on the given dataframe. Return only the python function without any execution and do not use print statement in the code block.
Below are two examples:
Instruction: count the number of entries whose category is Technology
Dataframe code for the first two records: import pandas as pd
data={'rank':[1.0, 2.0],'personName':['Elon Musk', 'Jeff Bezos'],'age':[50.0, 58.0],'finalWorth':[219000.0, 171000.0],'category':['Automotive', 'Technology'],'source':['Tesla, SpaceX', 'Amazon'],'country':['United States', 'United States'],…]}
df=pd.DataFrame(data)
Code: ```Python
# Define the function to count entries with category "Technology"
def target_function(dataframe):
  technology_entries_count = len(dataframe[dataframe['category'] == 'Technology])
  return technology_entries_count
```

Instruction: calculate the average of gold medals for the top 5 nations.
Dataframe code for the first two records: import pandas as pd
data={"Rank": ["1", "2"], "Nation": ["United States", "Jamaica"], "Gold": ["5", "4"], "Silver": ["6", "1"], "Bronze": ["5", "1"], "Total": ["16", "6"]}
df=pd.DataFrame(data)
Code: ```Python
# average number of gold medals for the top 5 nations in the dataframe
def target_function(dataframe):
  top_5_medals = dataframe["Gold"].astype(int).tolist()[:5]
  final_result = sum(top_5_medals) / 5
  return final_result
```
Now generate python code according to the following instruction.
Instruction: {instruction}
Dataframe code for the first two records: {table_df}
Code:

Figure 8: A prompt for a coding agent in the iterative planning module.

Given a question, the first two rows of a table, and four plans to solve the question, determine which plan is the most promising in terms of solving the question. Return only "The best path is {s}", where s the integer id of a plan.

Figure 9: A prompt LLM-as-judge to select the best plan.

ratings	title	text	author	date_stayed	offering_id	num_helpful_votes	date	id	via_mobile
{'service': 5.0, 'cleanliness': 5.0, 'overall'...	“Very nice experience for a country boy going ...	Being from a small town in Tennessee, I was ve...	{'username': 'Tucker124', 'num_reviews': 1, 'i...	October 2010	111492	2	2010-10-25 00:00:00+00:00	84800976	False
{'service': 5.0, 'cleanliness': 5.0, 'overall'...	“Great Courtyard”	I stayed at this courtyard for 2 nights . Ever...	{'username': 'bimbotrip', 'num_reviews': 1, 'i...	October 2009	108562	0	2009-10-14 00:00:00+00:00	46861760	False
...remaining 18 rows not shown...									

Question: List the 5 largest non-unique (they can be repeated) offering Ids.
Plan:

1. Group the data by 'offering_id' and **count the number of entries in each group.**
2. Sort the groups in descending order **based on the count.**
3. Select the top 5 'offering_id' values.
4. Return the list of the top 5 'offering_id' values.

Code:

```
Def target_function(dataframe):
    # Group the data by 'offering_id' and count the number of entries in each group
    offering_counts = dataframe.groupby('offering_id').size()
    # Sort the groups in descending order based on the count
    sorted_offering_counts = offering_counts.sort_values(ascending=False)
    # Select the top 5 'offering_id' values
    top_5_offering_ids = sorted_offering_counts.head(5).index.tolist()
    # Return the list of the top 5 'offering_id' values
    return top_5_offering_ids
```

Prediction: [82443, 84087, 93340, 93889, 94354]

Figure 10: An example of planning agent error.

Book Title	Author	Category	Price (TK)	Stock Status	Copies Left	Book Length (Pages)	Edition	Publication	Wished Users	Discount Offer	Ratings	Reviews
Sapiens A Brief History Of Humankind	Yuval Noah Harari	History and Tradition	1.258	In Stock	4.0	498	1st Published	Vintage Books (India)	386.0	Yes	39.0	16.0
The Intelligent Investor	Benjamin Graham	Business, Investment and Economics	1.598	In Stock	6.0	640	8th Impression	Harpercollins Publishers (India)	531.0	Yes	30.0	25.
...remaining 18 rows not shown...												

Question: Provide me with a list containing all the lengths of the books about computer science.
Plan:

1. Filter the table to include only rows where the 'Category' is **'Computer Science'.**
2. Extract the 'Book Length (Pages)' values from the filtered rows.
3. Compile these values into a list.
4. Return the list as the final answer.

Code:

```
Def target_function(dataframe):
    # Filter the table to include only rows where the 'Category' is 'Computer Science'
    filtered_df = dataframe[dataframe['Category'] == 'Computer Science']
    # Extract the 'Book Length (Pages)' values from the filtered rows
    book_lengths = filtered_df['Book Length (Pages)'].tolist()
    # Return the list of book lengths as the final answer
    return book_lengths
```

Prediction: []

Figure 11: An example of semantic matching error.

transaction_id	transaction_qty	store_id	store_location	product_id	unit_price	product_category	product_type	product_detail	Revenue	Month	Month_1	Weekday	Weekday_1	Hour
1	2	5	Lower Manhattan	32	3.00	Coffee	Gourmet brewed coffee	Ethiopia Rg	6,00 US$	1	Jan	7	Sun	7
2	2	5	Lower Manhattan	57	3.10	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	6,20 US$	1	Jan	7	Sun	7
…remaining 18 rows not shown…														

Question: List highest 3 different revenues a given transaction has yielded?
Plan:

1. Group the table by 'transaction_id' and calculate the sum of 'Revenue' for each group.
2. Sort the grouped table in descending order by the sum of 'Revenue'.
3. Select the top 3 unique 'Revenue' values from the sorted table.
4. Return the list of the top 3 different revenues."

Code:

```
Def target_function(dataframe):
    # Remove the dollar sign and comma from 'Revenue' and convert to float
    dataframe['Revenue'] = dataframe['Revenue'].str.replace(',', '').str.replace('US$', '').astype(float)
    # Group the dataframe by 'transaction_id' and calculate the sum of 'Revenue' for each group
    grouped_revenue = dataframe.groupby('transaction_id')['Revenue'].sum().reset_index()
    # Sort the grouped dataframe in descending order by 'Revenue'
    sorted_revenue = grouped_revenue.sort_values(by='Revenue', ascending=False)
    # Select the top 3 unique 'Revenue' values
    top_3_revenues = sorted_revenue['Revenue'].unique()[:3]
    # Return the list of the top 3 different revenues
    return top_3_revenues.tolist()
```

Prediction: [900.0, 700.0, 620.0]

Figure 12: An example of code generation error.

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	…29 columns unshown…
41	Yes	Travel_Rarely	1102	Sales	1	…
49	No	Travel_Frequently	279	Research & Development	8	…
…18 rows unshown…						

Question: List the 5 most common ages of our employees.
Answer: [38, 32, 29, 36, 22]
Prediction: [38, 29, 32, 41, 49]
Frequency of each Age (code executed): {32: 2, 38: 2, 29: 2, 41: 1, 49: 1, 37: 1, 33: 1, 27: 1, 59: 1, 30: 1, 36: 1, 35: 1, 31: 1, 34: 1, 28: 1, 22: 1, 53: 1}

Figure 13: An example of an ambiguous question

TL	#Q	#QT	#Row	#Col	#A				
					Bool	Ctg	Num	[Ctg]	[Num]
066_IBM_HR	39	10.0	1470	35	9	7	10	6	7
067_TripAdvisor	29	11.3	20000	10	9	1	13	3	3
068_WorldBank_Awards	34	12.4	239461	20	8	7	7	6	6
069_Taxonomy	35	12.5	703	8	9	7	8	8	3
070_OpenFoodFacts	29	10.8	9483	11	8	5	8	5	3
071_COL	36	12.7	121	8	8	7	8	6	7
072_Admissions	39	13.8	500	9	9	0	17	0	13
073_Med_Cost	32	10.5	1338	7	10	7	9	2	4
074_Lift	35	11.2	3000	5	9	4	10	6	6
075_Mortality	29	11.4	3000	5	9	4	10	6	6
076_NBA	36	13.0	8835	30	8	7	9	7	5
077_Gestational	31	13.0	1012	7	8	0	14	0	9
078_Fires	39	12.1	517	15	9	4	12	7	7
079_Coffee	38	12.5	149116	15	9	8	9	6	6
080_Books	41	12.9	40	13	8	5	14	7	7
DataBench_test	522	12.0	29066.4	13.3	129	74	156	72	91

Table 3: Statistics of DataBench test set. We present the names of each table in the TL column. #Q and #QT show and numbers of questions and the averaged numbers of question tokens (separated by white space) respectively. #Row and #Col show the averaged numbers of table rows and columns respectively. #A shows the numbers of answers. We categorize answer types into Boolean (Bool), Category (Ctg), Number (Num), a list of categorical values ([Ctg]) and a list of numerical values ([Num]) following Osés-Grijalba et al. (2025).