Augment before You Try: Knowledge-Enhanced Table Question Answering via Table Expansion

Anonymous ACL submission

Abstract

Table question answering is a popular task that assesses a model's ability to understand and interact with structured data. However, the given table often does not contain sufficient information to answer the question, necessitating the integration of external knowledge. Existing methods either convert both the table and external knowledge into text, which neglects the structured nature of the table; or they embed queries for external sources in the interaction with the table, which complicates the process. In this paper, we propose a simple yet effective method to integrate external information in a given table. Our method first constructs an augmenting table containing the missing information and then generates a SQL query over the two tables to answer the question. Experiments show that our method outperforms strong baselines on three table QA benchmarks.

1 Introduction

Tables are ubiquitous types of information sources that have attracted significant attention in the NLP community. Researchers have developed models to perform various tabular tasks, including table question answering (QA) (Pasupat and Liang, 2015; Chen et al., 2020c; Nan et al., 2022), table fact verification (Chen et al., 2020b; Aly et al., 2021), table-to-text generation (Parikh et al., 2020; Chen et al., 2020a; Nan et al., 2021), *etc.* A critical challenge in these tasks is that tables often lack sufficient information for the task at hand, which necessitates the integration of additional knowledge. For example, in Figure 1, to answer the question *'How many chords have a root not based on a sharp or flat note?'*, a model needs to have the knowledge of whether each root is based on a sharp or flat note, which is not provided in the table and can only be obtained from external sources.

Existing methods for integrating information from tables and external sources can be mainly categorized into two groups. The first method, exemplified by `Program-of-Thought` (Chen et al., 2023), linearizes the table into text and combines it with external knowledge in textual format (Xie et al., 2022; Chen, 2023). However, the linearized table no longer has the structured format, making it difficult to retrieve required values from the table and perform comparisons and calculations.

An alternative, `Binder` (Cheng et al., 2023), combines the symbolic language execution with large language models (LLMs). It interacts with the table through symbolic language like SQL, which maintains the structured format. Part of the SQL query is replaced with an LLM query that extracts knowledge from the LLM for further SQL execution. For instance, in Figure 1 (b), the method queries LLMs for whether each root is sharp or flat and uses the results as a filtering criterion in a SQL statement. However, it requires the model to learn to embed LLM queries in the standard SQL language, which differs substantially from the SQL statements the model has been trained on. As a result, it is more likely to generate syntactically wrong statements that lead to execution errors.

In this paper, we propose a simple yet effective method for combining external knowledge with a given table. As shown in Figure 1 (c), our method starts by analyzing the additional information required for answering the question. It then queries a knowledge source for the information and organizes the results in a tabular format. This newly created table *augments* the original table with additional information, and a SQL query is generated to obtain the answer from the two tables. Such an augment-then-generate pipeline eliminates the need to embed LLM queries in SQL statements while preserving the structured format of the table.

We evaluate our method on three table QA datasets that require different types of external knowledge (Chen et al., 2021; Zhu et al., 2021; Pasupat and Liang, 2015). Our method outperforms

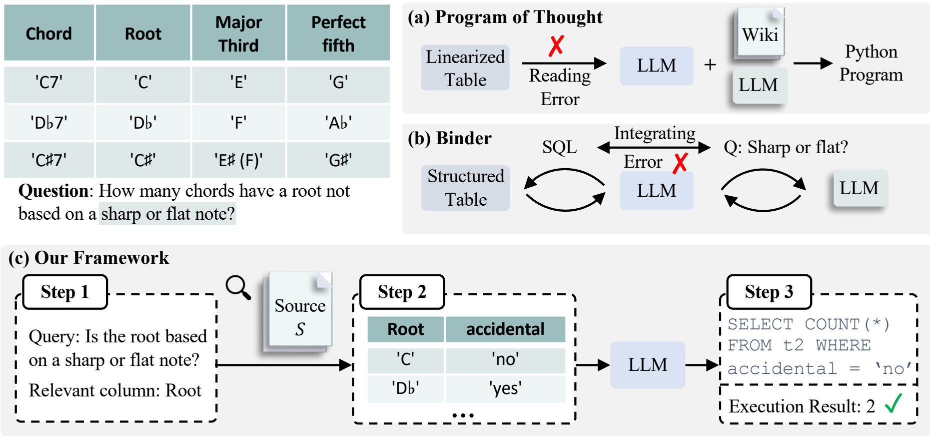

Figure 1: Comparison between `Program-of-Thought`, `Binder`, and our method.

or matches strong baselines on all datasets. Particularly, it demonstrates significant improvements over `Program-of-Thought` in questions with large tables or require complex tabular operations, and compared to `Binder`, it exhibits fewer execution errors and achieves better performance.

2 Related work

Table QA task combines structured data reasoning with text understanding. Traditional methods parse questions into executable commands to retrieve and process data from the table to obtain answers (Berant et al., 2013; Yin and Neubig, 2017; Zhong et al., 2017; Shaw et al., 2020; Yu et al., 2018). However, these methods require question-related information to present the table in a rigorous format, which is limited when applied to web tables that often do not have a clean schema. Recent works pre-train neural models on large-scale tabular data, and directly encode tables and generate answers in an end-to-end fashion (Liu et al., 2022; Xie et al., 2022; Herzig et al., 2020; Yin et al., 2020; Zhao et al., 2022; Deng et al., 2020). To reduce the training cost, some works leverage LLMs to read and reason over tables (Chen, 2023; Pourreza and Rafiei, 2023; Sui et al., 2024).

Although end-to-end methods excel on table QA benchmarks, their predictions lack interpretability and are not robust to input perturbations (Yang et al., 2022). For this reason, recent works combine LLMs with symbolic language execution. Particularly, Cheng et al. (2023) incorporates function calls to LLMs in SQL statements. Ye et al. (2023) decomposes the question and table into subproblems solvable by SQL queries. Chen et al. (2023) generates the reasoning process as Python programs. A recent work (Wang et al., 2024) dynamically updates the table in the reasoning process. They employ LLMs to iteratively generate operations such as selecting a subset of rows or adding a new column, and the final resulting table is fed to LLMs to generate the answer. However, their chain of operations is prone to error propagation, while our method retains the original table content and augments it with required information.

3 Methodology

3.1 Problem Formulation

Given a natural language question Q, a table T, and a knowledge source S, the task is to generate a correct answer for the question. Crucially, T might not contain all the necessary information to answer the question, which necessitates the use of S to obtain additional information. In this paper, we consider S to be either a relevant text document or an LLM that we can query.

3.2 Overall Framework

Our method contains three steps, as illustrated in Figure 1 (c). The detailed instructions and examples for each step are listed in Appendix A.

Step 1: Analyze question. An LLM is instructed to analyze the given question and table to determine what additional information is needed to answer the question. We instruct the LLM to first list out all the necessary information for answering Q. For each piece of information, it then determines if the information is present in T or not. The output of this step is a list of queries that can be later used to obtain additional information from S, or empty if no additional information is needed. For example, in Figure 1 (c), the model outputs *'Is the root based on a sharp or flat note?'*. Additionally, for information that needs to be obtained based on the table, the LLM will also specify which columns are needed, *e.g.,* it specifies that the query needs to be answered for each row in the 'Root' column.

Step 2: Construct augmenting table. Using output queries from step 1, the LLM then obtains corresponding information from the source S. Specifically, when S is a text document, this step is similar to the reading comprehension task where the LLM needs to extract answers to the queries from the document. When S is an LLM, this step resembles a QA task where the LLM needs to directly answer the query. Finally, the obtained information is organized into a separate table that can complement the existing table T. Figure 1 (c) shows an example where a new table of two columns is constructed. It is worth mentioning that this step is flexible and can be easily extended to other types of sources S.

Step 3: Generate SQL query. With the original and newly constructed tables, the LLM then generates a SQL query that can be executed to obtain the answer to the question. Importantly, the two tables contain sufficient information for answering Q, and the LLM can generate a standard SQL query, which is easier and more similar to its pre-training data.

4 Experiments

We evaluate our method on table QA benchmarks, focusing on two types of questions that might require external knowledge from different sources.

- **Open-domain knowledge** where external information comes from an open domain. We use the embedded knowledge in LLMs as the source.
- **Closed-domain knowledge** where all information is within a given table and a text document containing all related external knowledge.

We will discuss the common experiment settings in Section 4.1 and individual experiments for each type in Sections 4.2 and 4.3 respectively.

4.1 Experiments Setup

Implementation details. We prompt an LLM with detailed instructions and in-context examples for all three steps in our method. To feed the table to the LLM for question analysis (Step 1) and generating SQL queries (Step 3), we linearize the table by concatenating columns with special tokens (*e.g.,* '|') following previous works (Chen et al., 2023). We use `GPT-3.5-turbo-1106` as the backbone LLM and greedy decoding (*i.e.,* temperature is 0) for our method and all baselines. For a fair comparison, we use the same number of in-context examples as baselines (details in Appendix A).

Baselines. We compare with five LLM-based baselines. ❶ `End-to-End` that directly outputs the answer given the table, question, and optionally the text document. ❷ `Table-CoT` (Chen, 2023) that uses the chain-of-thought prompting (Wei et al., 2022) to additionally output the reasoning chain. ❸ `Dater` (Ye et al., 2023), ❹ `Binder` (Cheng et al., 2023), and ❺ `Program-of-Thought` (`PoT`) (Chen et al., 2023) that combine LLMs with symbolic language execution (details in Section 2). Particularly, since `Binder` does not generate the reasoning chain, we include an improved variant with chain-of-thought prompting, denoted as `Binder+CoT`.

Metrics. We use exact match rate (EM) between predicted and ground-truth answers as the metric and use the same evaluation code across methods.

4.2 Open-Domain Knowledge

Datasets. We evaluate on WikiTQ dataset (Pasupat and Liang, 2015), which requires complex table reasoning for the question. According to Shi et al. (2020), around 20% of WikiTQ questions are not answerable by SQL queries, which are likely to require additional knowledge not present in the table. We test all methods on the full test set, containing 4344 samples.

Results. Table 1 presents the EM. There are two observations from the table. First, methods that involve program execution are generally better than those that do not, highlighting the value of accurate data retrieval or processing. Second, our method achieves the best performance, showing its effectiveness. To further evaluate scalability across table sizes, Figure 2 plots the performance breakdown by the number of tokens in the table. As can be observed, our method and `Binder+CoT` are the only methods that maintain performance on large tables, whereas methods that rely on LLMs to extract information from linearized tables such as `Table-CoT` and `PoT` suffer significant performance degradation on large tables. This illustrates the advantage of SQL queries when interacting with the table.

Comparison with `Binder+CoT`. To further verify whether our augment-then-generate pipeline leads to easier and more accurate SQL generation over the best-performing baseline `Binder+CoT` (hereafter `Binder`), we compare the two methods on the subset of questions not solvable by pure SQL identified by Shi et al. (2020), which rely more on the integration of external knowledge. Fig-

	Test EM
End-to-End	50.78
Table-CoT (Chen, 2023)	52.42
PoT (Chen et al., 2023)	53.02
Dater (Ye et al., 2023)	46.89
Binder (Cheng et al., 2023)	35.45
Binder+CoT	52.09
Ours	**55.69**

Table 1: Exact match on WIKITQ test set. Methods in the bottom panel involve program execution.

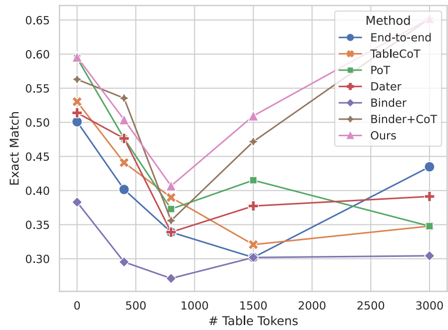

Figure 2: Performance grouped by table length.

ure 3 shows the EM and percentage of execution errors, where our method demonstrates a more pronounced improvement. To better pinpoint the cause of performance difference, we add a post-processing step for Binder, where we extract the LLM queries from the SQL statement generated by Binder, query LLMs for desired information and add it as a new column in the original table, and re-generate a standard SQL (without LLM queries) based on the augmented table. This variant (dubbed Binder-separate) improves the EM and reduces execution errors over Binder, which validates our hypothesis that combining LLM queries with SQL complicates the generation, leading to more syntax errors in generated programs. Notably, our method still incurs fewer execution errors than Binder-separate, which is likely due to the fact that our method generates more augmentations for the table, thus reducing the complexity of required SQL (see Appendix C.1 for details and examples).

In Appendix B, we also compare our method with a recent work Chain-of-Table (Wang et al., 2024). Results show that our method achieves 1.85 higher EM when using GPT3.5-0613 as the backbone LLM, demonstrating its effectiveness despite being simpler and not requiring sequential operations. Please refer to Appendix B for details.

4.3 Closed-Domain Knowledge

Datasets. We evaluate on TATQA (Zhu et al., 2021) and FinQA (Chen et al., 2021). Questions in these datasets involve a table and a financial report, and the answer often requires arithmetic operations in addition to table understanding ability. We filter the datasets to only include questions that require both table and report to answer (details in Table 5).

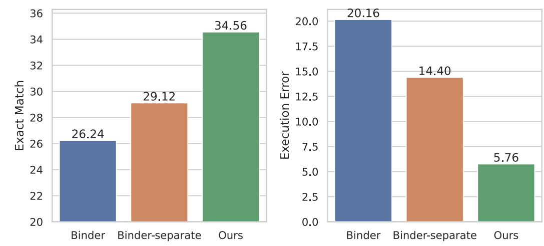

Figure 3: Comparison between our method and Binder.

	TATQA	FINQA
End-to-End	35.50	34.18
Table-CoT (Chen, 2023)	34.91	39.87
PoT(Chen et al., 2023)	61.14	**54.43**
Ours	**63.12**	53.80

Table 2: Exact match on TATQA and FINQA.

Results. Table 2 presents the results. Binder and Dater are not included because the original paper did not evaluate on these datasets and extension to this setting requires substantial modification. There are two observations. First, our method and PoT significantly outperform the other two baselines that do not involve program executions, which shows the benefits of leveraging programs when questions require arithmetic calculations. Second, although the input tables are much smaller, which is beneficial for PoT, our method is on par with PoT on FINQA and outperforms it by 2 EM on TATQA. A further performance breakdown by the number of table cells required to answer a question in Figure 4 shows that our method is more effective on questions that require information from multiple cells, indicating that our method is more likely to generalize to complex questions. Furthermore, it is easier to locate and correct errors made by our method as it only requires inspection of the generated SQL queries, whereas PoT requires checking the whole table contents (see examples in Appendix C.2).

5 Conclusion

We propose a simple method that augments a table by creating a new table that contains information from external sources. The LLM then generates a SQL query to answer the question. Experiments on three table QA benchmarks show that our method outperforms or matches strong baselines.

6 Limitation

There are several limitations in this work that need to be further improved. First, our framework relies on the LLM's ability to generate correct SQL statements. If the LLM has limited SQL generation ability, such as `Llama2` in Appendix B, the performance of our method will be affected. In addition, we only evaluate our method on integrating external knowledge from two different sources. The generalizability of our method to other knowledge sources remains to be assessed.

7 Potential Risks and Use of Data

In this paper, we propose a method for the table QA task that combines LLMs with SQL queries. Each step of our method is interpretable, which allows users to easily verify the correctness of each step. Thus the potential risks of our method can be considerably reduced. However, it is also important to note that our method relies on LLMs to obtain additional information. Particularly, when the LLM is used as the external source, it might encounter the hallucination issue, where the LLM generates wrong information for the question. It is thus crucial to not fully trust the output from the LLM and compare it with reliable information sources to check the correctness of augmented information.

The datasets used in this paper are downloaded from the official websites. All datasets are under CC-BY-4.0 license and are consistent with their intended use. Table 5 lists the statistics of employed datasets.

References

Rami Aly, Zhijiang Guo, Michael Sejr Schlichtkrull, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, Oana Cocarascu, and Arpit Mittal. 2021. The fact extraction and VERification over unstructured and structured information (FEVEROUS) shared task. In *Proceedings of the Fourth Workshop on Fact Extraction and VERification (FEVER)*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544.

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

Wenhu Chen, Jianshu Chen, Yu Su, Zhiyu Chen, and William Yang Wang. 2020a. Logical natural language generation from open-domain tables. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020b. Tabfact: A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations*.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020c. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages. In *The Eleventh International Conference on Learning Representations*.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. Turl: Table understanding through representation learning.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: Table pre-training via learning a neural SQL executor. In *International Conference on Learning Representations*.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, Mutethia Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, Dragomir Radev, and Dragomir Radev. 2022. FeTaQA: Free-form table question answering. *Transactions of the Association for Computational Linguistics*.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Mutethia Mutuma, Yasin Tarabar, Ankit Gupta, Tao Yu, Yi Chern Tan, Xi Victoria Lin, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. 2021. DART: Open-domain structured data record to text generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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)*.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv: 2304.11015*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to SQL queries. In *Findings of the Association for Computational Linguistics: EMNLP 2020*.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat 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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2022. UnifiedSKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*.

Jingfeng Yang, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel, and Shachi Paul. 2022. TableFormer: Robust transformer modeling for table-text encoding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decompose evidence and questions for table-based reasoning.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

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*.

Yilun Zhao, Linyong Nan, Zhenting Qi, Rui Zhang, and Dragomir R. Radev. 2022. Reastap: Injecting table reasoning skills during pre-training via synthetic reasoning examples. *Conference on Empirical Methods in Natural Language Processing*.

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. 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)*.

A Implementation Details

For all methods on all three datasets, we use the greedy decoding for generation, *i.e.,* temperature equals 0. Table 3 lists other generation parameters of our method. Table 5 shows the statistics of datasets used in this paper.

A.1 Open-domain Knowledge

For the open-domain knowledge setting on WIKITQ, since our method generates queries that will be asked for every single row in one or more columns, the constructed augmenting table will always have the same number of rows as the original table. For simplicity, we directly join the two tables based on the row index before feeding them to LLMs to generate the SQL statement in step 3. In other words, the newly constructed table is joined on the original table as additional columns, and the SQL statement will be generated based on the joined table. Figures 11 and 12 show the detailed instruction used for this step and a demonstration of the in-context example. For step 2, we use the same instruction and in-context examples as Cheng et al. (2023) to query LLMs for required information. An example is shown in Figure 13. For step 3, we provide LLMs with in-context examples along with a one-sentence instruction, as illustrated in Figure 14. We use the evaluation code in Cheng et al. (2023) to calculate EM for all methods.

A.2 Closed-domain Knowledge

For the closed-domain setting on TATQA and FINQA, we feed both the text document and the table to the LLM to provide enough context. To save the inference cost, we merge steps 1 and 2 together such that the model analyzes the required additional information and then extracts them from the document in a single run. We instruct the model to extract information in a JSON format that can be easily organized into a table. Figures 15 and 16 show the detailed instruction used and a demonstration of the in-context example on TATQA, and Figures 18 and 19 show the same for FINQA. For step 3, we provide the original table and the newly constructed table if available to LLMs. Figures 17 and 20 show a demonstration of the in-context examples on TATQA and FINQA respectively.

To select questions for evaluation, we only use those that require both the table and the document. Specifically, for TATQA, we select questions that have `answer_from=table-text`, and for FINQA, we select those whose ground truth evidence contains at least one table row and one document sentence. We follow Chen et al. (2023) to calculate the EM.

	WIKITQ	TATQA	FINQA
top_p	1.0	1.0	1.0
max_output_tokens	512	512	512
num_shots	8	8	4

Table 3: Parameters for our greedy generation (sections 4.2 and 4.3).

	GPT3.5		Llama2	
	Augmentation generation	SQL generation	Augmentation generation	SQL generation
temperature	0.6	0.4	0.8	0.4
top_p	1.0	1.0	1.0	1.0
sampling_n	3	2 or 4	4	3 or 4
max_output_tokens	512	512	256	256
num_shots	8	8	8	8

Table 4: Generation parameters for our ensemble model on WIKITQ (Appendix B). Augmentation generation and SQL generation correspond to the step 1 and 3 in our method.

B Comparison with `Chain-of-Table`

We additionally compare with `Chain-of-Table` (Wang et al., 2024) on WIKITQ. Since their implementation is not available at the submission time of this paper, we use the same dataset and backbone LLMs as theirs and directly compare with the numbers reported in their paper. Specifically, we use `GPT-3.5-turbo-16k-0613` and `Llama2-13b-chat` (Touvron et al., 2023) as backbone LLMs and evaluate on the full test set of WIKITQ. Since their sequential operations require multiple queries for LLMs, we consider the majority vote of execution results from N SQL queries as our final prediction. To generate these SQL

	WIKITQ	WIKITQ SQL unsolvable	TATQA	FINQA
# questions	4344	625	507	158
Split	Test	Dev	Dev	Test
# table rows	25.4	28.0	9.7	6.8
# table tokens	571.7	685.7	119.1	86.2
Knowledge source S	LLMs	LLMs	Document	Document

Table 5: Summary of the datasets used in this paper.

	# generated samples	EM
GPT3.5		
Binder	50	56.74
Chain-of-Table	≤ 25	59.94
Ours (6 SQLs)	11.4	61.05
Ours (12 SQLs)	17.4	**61.79**
Llama2		
Binder	50	30.92
Chain-of-Table	≤ 25	**42.61**
Ours (12 SQLs)	19.82	34.00
Ours (16 SQLs)	23.82	35.34

Table 6: Exact match on full WIKITQ test set. # generated samples denotes the total number of generated samples to answer one question.

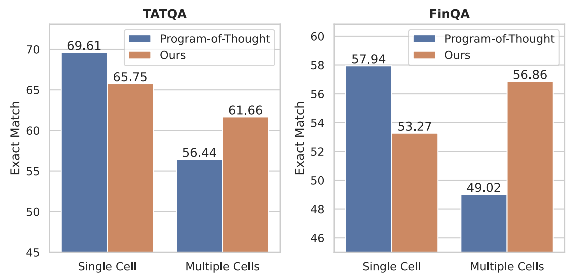

Figure 4: Performance decomposition by the number of table cells needed to answer the question.

queries, we sample m different outputs for step 1 (*i.e.,* m different augmentations), and for each augmentation, we sample k SQL queries. The total number of generated samples for each question is $m + \alpha m + mk$, where α is the percentage of step 1 outputs that actually need additional information. Table 4 lists the parameters for generation.

The results are shown in Table 6. As can be observed, our method outperforms Chain-of-Table and Binder when using GPT3.5 as the backbone LLM, despite using fewer LLM queries. When using Llama2, Chain-of-Table achieves better performance than Binder and our method. We hypothesize that the performance difference is due to the limited SQL generation ability of Llama2. An important difference is that Chain-of-Table feeds the final table to LLMs and directly asks LLMs to generate the answer, whereas Binder and our method prompt LLMs to generate SQL queries and execute to get the answer, which is affected more when the LLM has limited SQL generation ability. In fact, the generated SQL of our method contains 32.9% of execution errors when using Llama2 as the LLM, compared to that of 8.7% when using GPT3.5. However, our method still outperforms Binder on Llama2, demonstrating the benefits of our augment-then-generate pipeline.

C Additional Examples

C.1 Comparison with Binder

In this section, we elaborate on the comparison between our method, Binder, and Binder-separate. In Figure 3, it can be observed that our method achieves better performance and exhibits fewer execution errors than Binder. Moreover, Binder-separate, which separates the SQL generation and LLM queries in Binder, reduces its execution errors, validating our hypothesis that integrating LLM queries in SQL generation could lead to more syntax errors. Figures 5 and 6 show two examples where Binder encounters execution errors when trying to generate a SQL statement with LLM queries, whereas our method and Binder-separate correctly generate SQL statements to answer the question.

Our method also incurs fewer execution errors than Binder-separate, which can be ascribed to the fact that our method generates more augmentations for the table, which significantly reduces the complexity of required SQL statements. Figure 7 illustrates one such example, where Binder-separate gets errors because the required information is missing from the table, whereas our method correctly answers the question based on the augmented table. In fact, our method generates augmentations for 72.3% of the questions, while Binder only includes LLM queries for 6.1% of the questions, showing that our method also benefits the augmentation of additional information.

C.2 Comparison with PoT

We now provide more examples for the comparison between our method and PoT. Figure 4 shows the performance breakdown by the number of cells required to answer the question. Based on the figure, our method is more effective on questions that require multiple table cells for the answer. Figures 8 and 9 show two such examples, where our method selects the correct values from the table to perform calculations, but PoT retrieves wrong values from the table, despite generating programs with correct logic. According to Chen et al. (2023), this type of value grounding errors take up 47% of the errors made by PoT. Moreover, correcting these errors requires manual efforts to look into the contents of the table, which is time-consuming when the table is large.

On the contrary, Figure 10 shows an example question that only requires a single cell from the table. PoT correctly selects the answer but our method selects the value in the wrong column. However, correcting this error requires only manual inspection of the generated SQL statement, which is much more efficient than checking the whole table contents.

row_id	first issued	serials format	serials issued
0	1955	a-12345	none
1	1958	abc-123	aaa-1 to bzz-999
		…	
9	2008	abc1234	aaa0001 to present

Question: name the year of the license plate that has the largest amount of alphanumeric digits

Binder:

```
SELECT `first issued` FROM t1 ORDER BY
LENGTH(QA("map@alphanumeric digits"; CONCAT(`serial
format`, `serials issued`))) DESC LIMIT 1
(sqlite3.OperationalError) near "CONCAT": syntax error
```

Ours:

Step 1: How many alphanumeric digits are there in the serial format?

Step 2: Augmented table:

row_id	first issued	serials format	serials issued	alphanumeric_digits
0	1955	a-12345	none	6
1	1958	abc-123	aaa-1 to bzz-999	6
		…		
9	2008	abc1234	aaa0001 to present	7

Step 3:

```
SELECT `first issued` FROM t1 ORDER BY
`alphanumeric_digits` DESC LIMIT 1
```

Execution result: 2008 ✓

Figure 5: An example question in WikiTQ. Binder generates a SQL statement that queries LLMs for unsolvable parts. However, the statement leads to an execution error. Our method augments the table with an additional column and correctly generates a SQL statement to answer the question.

row_id	Time created	map or catalog
0	1000 bc	wu xian star map
1	168 bc	5 star prediction device
	…	
133	193 bc	han comet diagrams

Question: what is the difference between the five star prediction device's date of creation and the han comet diagrams' date of creation?

Binder:

```
SELECT ABS(QA("map@date of creation"; `time created`
WHERE `map or catalog` = '5 star prediction device')
- QA("map@date of creation"; `time created` WHERE
`map or catalog` = 'han comet diagrams'))
(sqlite3.OperationalError) near "`time created`":
syntax error
```

Binder-separate:

Added column: None

SQL:

```
SELECT ABS(
    (SELECT `time created` FROM t1 WHERE `map or catalog`
= '5 star prediction device') -
    (SELECT `time created` FROM t1 WHERE `map or catalog`
= 'han comet diagrams'))
```

Execution result: 25 ✓

Ours:

Step 1: None

Step 2: None

Step 3:

```
SELECT ABS(
    (SELECT `time created` FROM t1 WHERE `map or catalog`
= '5 star prediction device') -
    (SELECT `time created` FROM t1 WHERE `map or catalog`
= 'han comet diagrams'))
```

Execution result: 25 ✓

Figure 6: An example question in WikiTQ. Binder generates a SQL statement that queries LLMs. However, the statement leads to an execution error. Our method and Binder-separate correctly generate a pure SQL statement to answer the question.

row_id	pos	driver	points
0	1	fernando alonso	10.0
1	2	jarno trulli	8.0
		…	
19	ret	patrick friesacher	nan

Question: jarno trulli was not french but what nationality?

Binder:

```
SELECT 'jarno trulli' FROM t1 WHERE `driver` =
'jarno trulli'
```

Execution result: jarno trulli ✗

Binder-separate:

Added column: None

SQL:

```
SELECT DISTINCT `driver`, `nationality` FROM t1 WHERE
`driver` = 'jarno trulli'
(sqlite3.OperationalError) no such column: nationality
```

Ours:

Step 1: What is the nationality?

Step 2: Augmented table:

row_id	pos	driver	points	nationality
0	1	fernando alonso	10.0	spanish
1	2	jarno trulli	8.0	italian
		…		
19	ret	patrick friesacher	nan	austrian

Step 3:

```
SELECT `nationality` FROM t1 WHERE `driver` =
'jarno trulli'
```

Execution result: italian ✓

Figure 7: An example question in WikiTQ. Binder generates a SQL statement without syntax error, but it does not query LLMs for additional knowledge, resulting in a wrong answer. Since the original SQL statement generated by Binder does not augment the table with additional information, the question is not answerable using pure SQL, leading to an execution error in Binder-separate. Our method augments the table and correctly answers the question.

row_id		2018	2017	2016
0	balance at beginning of fiscal year	148.9	166.8	106.6
1	additions related to purchase accounting (1)	3.4	7.7	16.5
	…			
8	balance at end of fiscal year	127.1	148.9	166.8

Question: by what percent did the balance of tax benefits increase between the beginning of 2016 and the end of 2018?

Program-of-Thought:

```
balance_beginning_2016 = 166.8
balance_end_2018 = 127.1
percent_increase = ((balance_end_2018 -
balance_beginning_2016) / balance_beginning_2016) * 100
ans = percent_increase
```

Execution result: -23.80 ✗

Ours:

```
SELECT
    ((SELECT `2018` FROM t1 WHERE `row_id` = 8)
    -
    (SELECT `2016` FROM t1 WHERE `row_id` = 0))
    /
    (SELECT `2016` FROM t1 WHERE `row_id` = 0) * 100
```

Execution result: 19.23 ✓

Figure 8: An example question in FinQA that requires two table cells to answer. PoT retrieves the wrong value (highlighted in red) from the table, despite generating a program with correct logic. Identifying the error requires looking into the table contents manually. Our method correctly selects the values and answers the question.

row_id		2019	2018
0	deferred tax assets:	none	none
12	total deferred tax assets	83615	26062
	…		
18	net deferred tax assets (liabilities)	48218	-1221

Question: What was the percentage change in the Net deferred tax assets (liabilities) between 2018 and 2019?

Program-of-Thought:

```
net_deferred_tax_assets_2018 = 26062
net_deferred_tax_assets_2019 = 83615
ans = (net_deferred_tax_assets_2019 -
net_deferred_tax_assets_2018) /
abs(net_deferred_tax_assets_2018) * 100
```

Execution result: 220.83 ✗

Ours:

```
SELECT
    ((CAST(`2019` AS REAL) - CAST(`2018` AS REAL)) /
CAST(`2018` AS REAL)) * 100 AS percentage_change
FROM t1
WHERE `row_id` = 18
```

Execution result: -4049.06 ✓

Figure 9: An example question in TATQA that requires two table cells to answer. PoT retrieves the wrong value (highlighted in red) from the table, despite generating a program with correct logic. Our method correctly selects the values and answers the question.

row_id		july 27, 2019 (1)	2019 vs. 2018 (variance in dollars)
0	revenue:	none	none
12	product	39005	2296
	…		
5	total	51904	2574

Question: What was the product revenue variance in dollars for 2019 vs 2018?

Program-of-Thought:

```
product_revenue_variance_2019_vs_2018 = 2296
```

Execution result: 2296 ✓

Ours:

```
SELECT `july 27, 2019 (1)` FROM t1 WHERE `row_id` = 1
```

Execution result: 39005 ✗

Figure 10: An example question in TATQA that requires a single table cell to answer. PoT correctly retrieves the value from the table. Our method mistakenly selects the value. However, the error is easy to be spotted and corrected by inspecting the SQL statement.

```
Task Description:
Your task is to prepare a table for SQL query generation in order to answer a specific
question. This may require modifying the table by adding extra columns. These new
columns are created based on natural language questions, with each question applied
individually to every row in the existing columns. The goal is to transform existing
data into a format that's suitable for SQL operations, or to incorporate additional
information into the table.

Procedure:
1. Evaluate the Table and Question: Assess if the table, in its current format, is
suitable for generating a SQL query to answer the given question.
2. Determine Additional Columns:
   - If the table is already suitable for the SQL query, simply output "None"
   - If the table requires modifications, identify and define the necessary changes.
Specifically, add new columns where each row's value is derived from a natural language
question applied to the relevant columns. Use the format:
   `new_column` = @("question"; [relevant_columns]),
   where `question` is the question asked for each row to derive the new column's
contents, and `relevant_columns` are the existing columns that provide the information
needed for the question.

Response Format:
Begin your response with "Transformation:" and include:
- Solution outline: Describe a step-by-step reasoning chain of how to answer the
question.
- Further analysis: Determine if modifications are required for each step.
- Final output: List each required additional column in the specified format, each on a
new line. If no modifications are needed, output "None".
```

Figure 11: System prompt used for augmentation generation (Step 1) on WIKITQ.

```
Title: 2007 New Orleans Saints season
CREATE TABLE t1(
    row_id int,
    date text,
    game site text,
    result/score text)
/*
3 example rows:
SELECT * FROM t1 LIMIT 3;
row_id    date         game site                result/score
0         2007-9-6     rca dome                 l 41-10
1         2007-9-16    raymond james stadium    l 31-14
2         2007-9-24    louisiana superdome      l 31-14
*/

Q: what number of games were lost at home?
Transformation:
Solution outline:
1. Find the losing games.
2. Find the games at home.
3. Count the number of games that satisfy both conditions.
Further analysis:
For step 1, we need information in `result/score` column. We need to parse if it's a win
 or loss. We will add a column called `is_loss`.
For step 2, we need information in `game site` column. We need additional information on
 whether it's a home game or not. We will add a column called `is_home_game`.
Step 3 can be done with a SQL query.
Final output:
`is_loss` = @("Is it a loss?"; [result/score])
`is_home_game` = @("Is it the home court of New Orleans Saints?"; [game site])
```

Figure 12: A demonstration of the in-context example used for augmentation generation (Step 1) on WIKITQ.

```
Give a database as shown below:
Table: 1963 International Gold Cup
/*
row_id  driver
0    jim clark
1    richie ginther
2    graham hill
3    jack brabham
4    tony maggs
*/
Q: Answer question "What is his/her country?" row by row.
Output:
/*
row_id  driver
0    jim clark        scotland
1    richie ginther   united states
2    graham hill      england
3    jack brabham     australia
4    tony maggs       south africa
*/
```

Figure 13: A demonstration of the in-context example used for querying additional information (Step 2) from LLMs on WIKITQ.

````
Read the following table and write a SQL query to answer the question:
Title: 2007 New Orleans Saints season
CREATE TABLE t1(
    row_id int,
    date text,
    game site text,
    result/score text,
    is_loss text,
    is_home_game text)
/*
3 example rows:
SELECT * FROM t1 LIMIT 3;
row_id    date        game site                result/score    is_loss    is_home_game
0         2007-9-6    rca dome                 l 41-10         yes        no
1         2007-9-16   raymond james stadium    l 31-14         yes        no
2         2007-9-24   louisiana superdome      l 31-14         yes        yes
*/

Q: what number of games were lost at home?
SQL: To answer the question, we need following steps:
1. Find the losing games by `is_loss` column.
2. Find the games at home by `is_home_game` column.
3. Count the number of games that satisfy both conditions.
Final SQL query:
```
SELECT COUNT(*) FROM t1 WHERE `is_loss` = 'yes' AND `is_home_game` = 'yes'
```
````

Figure 14: A demonstration of the in-context example used for SQL generation (Step 3) on WIKITQ.

```
Task Description:
You are tasked with analyzing a provided table and an accompanying report to answer a
specific question. This involves assessing whether the table contains all necessary
information for answering the question. If additional information is needed, you must
extract this from the report and create a supplementary table. Your primary focus is on
the analysis and information extraction process, which will facilitate in forming a SQL
query to answer the question.

Procedure:
1. Assess the Given Table and Question: Determine whether the provided table contains
all the required information to answer the question.
2. Extract Information for Additional Table Creation:
   - If the existing table is sufficient, simply output "None"
   - If the existing table lacks essential information, extract the required data from
the report in the following JSON format: `{"column_name": [value1, ...], ...}`

Each example is given in the following structure:
- Report: Contents of the report that may contain additional information.
- Tables: Contents of the table, with columns separated by " | " and rows by "\n".
- Question: The specific question that needs to be answered.

Response Format:
Begin your response with "Analysis:" and include:
- Solution outline: Describe the step-by-step outline for answering the question.
- Further analysis: Determine whether each step's information is available in the
existing table or needs to be extracted from the report.
- Final output: Extract necessary information from the report in JSON format as
described above; if no additional information is needed, output "None".

Notes:
- You may extract information with any number of columns and rows. However, all columns
should have the same number of values.
- Make the JSON self-explanatory. Use descriptive column names, add context where needed
, and include units in column names to prevent ambiguity.
- Avoid creating columns with empty or NaN values.
```

Figure 15: System prompt used for constructing augmenting table (Steps 1 and 2) on TATQA.

```
Report:
NOTE 5 - PROPERTY AND EQUIPMENT
The Company owned equipment recorded at cost, which consisted of the following as of
December 31, 2019 and 2018:
Depreciation expense was $80,206 and $58,423 for the years ended December 31, 2019 and
2018, respectively
Tables:
row_id | filledcolumnname | 2019 | 2018
0 | computer equipment | 137763 | 94384
1 | furniture and fixtures | 187167 | 159648
2 | subtotal | 324930 | 254032
3 | less accumulated depreciation | 148916 | 104702
4 | property and equipment, net | 176014 | 149330

Question: What is the ratio of depreciation expense to accumulated depreciation of
property and equipment in 2019?
Analysis:
Solution outline:
1. Find the amount of depreciation expense and accumulated depreciation of property and
equipment in 2019.
2. Calculate the ratio.
Further analysis:
For step 1, the accumulated depreciation is mentioned in the table in row 3. But the
depreciation expense is missing from the table. So we need to extract it from the report
.
Step 2 can be done with a SQL query.
Final output:
{"depreciation_expense_2019": ["$80,206"]}
```

Figure 16: A demonstration of the in-context example used for constructing augmenting table (Steps 1 and 2) on TATQA.

```
Report:
NOTE 5 - PROPERTY AND EQUIPMENT The Company owned equipment recorded at cost, which
consisted of the following as of December 31, 2019 and 2018: Depreciation expense was
$80,206 and $58,423 for the years ended December 31, 2019 and 2018, respectively
Tables:
CREATE TABLE t1(
    row_id int,
    filledcolumnname text,
    2019 int,
    2018 int)
/*
All rows of the table:
SELECT * FROM t1;
row_id  filledcolumnname        2019    2018
0       computer equipment      137763  94384
1       furniture and fixtures  187167  159648
2       subtotal        324930  254032
3       less accumulated depreciation   148916  104702
4       property and equipment, net     176014  149330
*/

CREATE TABLE t2(
    row_id int,
    depreciation_expense_2019 int)
/*
All rows of the table:
SELECT * FROM t2;
row_id  depreciation_expense_2019
0       80206
*/

Q: What is the ratio of depreciation expense to accumulated depreciation of property and
 equipment in 2019?
SQL: Reasoning process:
We need following steps to answer the question:
1. Get the depreciation expense in 2019 from t2.
2. Get the accumulated depreciation in 2019 from t1, which is in row 3.
3. Calculate the ratio.
Final SQL query:
\`\`\`
SELECT
    (SELECT `depreciation_expense_2019` FROM t2 WHERE `row_id` = 0) /
    CAST((SELECT `2019` FROM t1 WHERE `row_id` = 3) AS REAL)
    AS depreciation_ratio
FROM t1
LIMIT 1
\`\`\`
Units: ""
```

Figure 17: A demonstration of the in-context example used for SQL generation (Step 3) on TATQA.

```
Task Procedure:
1. Assess the Given Table and Question: Determine whether the provided table contains
all the required information to answer the question.
2. Extract Missing Information from Report:
   - If the existing table is sufficient, simply output "None"
   - If the existing table lacks essential information, extract the required data from
the report in the following JSON format: `{"column_name": [value1, ...], ...}`

Each example is given in the following structure:
- Report: Contents of the report that may contain additional information.
- Tables: Contents of the table, with columns separated by " | " and rows by "\n".
- Question: The specific question that needs to be answered.

Response Format:
Begin your response with "Analysis:" and include:
- Solution formula: Write a formula to calculate the answer.
- Further analysis: Determine for each variable in the formula whether it is available
in the table or needs to be extracted from the report.
- Final output: For variables not in the table, extract them from report in JSON format
as described above; if all variables are in the table, output "None".

Notes:
- Make the JSON self-explanatory. Use descriptive column names and include units in
column names to prevent ambiguity.
```

Figure 18: System prompt used for constructing augmenting table (Steps 1 and 2) on FINQA.

```
Report:
purchases of equity securities 2013 during 2014 , we repurchased 33035204 shares of our
common stock at an average price of $ 100.24 .
[b] effective january 1 , 2014 , our board of directors authorized the repurchase of up
to 120 million shares of our common stock by december 31 , 2017 .
Tables:
row_id | period | total number ofsharespurchased[a] | averageprice paidpershare | total
number of sharespurchased as part of apublicly announcedplan or program [b] | maximum
number ofshares that may yetbe purchased under the planor program [b]
0 | oct . 1 through oct . 31 | 3087549 | 107.59 | 3075000 | 92618000
1 | nov . 1 through nov . 30 | 1877330 | 119.84 | 1875000 | 90743000
2 | dec . 1 through dec . 31 | 2787108 | 116.54 | 2786400 | 87956600
3 | total | 7751987 | 113.77 | 7736400 | n/a

Question: what percent of the share repurchases were in the fourth quarter?
Analysis:
Solution formula:
share_repurchase_fourth_quarter / share_repurchase_whole_year
Further analysis:
share_repurchase_fourth_quarter is in row 3 of the table
share_repurchase_whole_year is not in the table, so we need to extract it from the
report
Final output:
{"share_repurchase_whole_year": [33035204]}
```

Figure 19: A demonstration of the in-context example used for constructing augmenting table (Steps 1 and 2) on FINQA.

```
Report:
purchases of equity securities 2013 during 2014 , we repurchased 33035204 shares of our
common stock at an average price of $ 100.24 .
[b] effective january 1 , 2014 , our board of directors authorized the repurchase of up
to 120 million shares of our common stock by december 31 , 2017 .
Tables:
CREATE TABLE t1(
    row_id int,
    period text,
    total number ofsharespurchased[a] int,
    averageprice paidpershare real,
    total number of sharespurchased as part of apublicly announcedplan or program [b]
    int,
    maximum number ofshares that may yetbe purchased under the planor program [b] text)
/*
All rows of the table:
SELECT * FROM t1;
row_id  period  total number ofsharespurchased[a]       averageprice paidpershare
total number of sharespurchased as part of apublicly announcedplan or program [b]
maximum number ofshares that may yetbe purchased under the planor program [b]
0       oct . 1 through oct . 31        3087549 107.59  3075000 92618000
1       nov . 1 through nov . 30        1877330 119.84  1875000 90743000
2       dec . 1 through dec . 31        2787108 116.54  2786400 87956600
3       total   7751987 113.77  7736400 n/a
*/

CREATE TABLE t2(
    row_id int,
    share_repurchase_whole_year int)
/*
All rows of the table:
SELECT * FROM t2;
row_id  share_repurchase_whole_year
0       33035204
*/

Q: what percent of the share repurchases were in the fourth quarter?
SQL:
Solution formula:
share_repurchase_fourth_quarter / share_repurchase_whole_year
Further analysis:
share_repurchase_fourth_quarter is in row 3, column `total number ofsharespurchased[a]`
of t1
share_repurchase_whole_year is in row 0, column `share_repurchase_whole_year` of t2
Final SQL query:
```
SELECT
    CAST((SELECT `total number ofsharespurchased[a]` FROM t1 WHERE `row_id` = 3) AS REAL
) /
    (SELECT `share_repurchase_whole_year` FROM t2 WHERE `row_id` = 0) * 100
```
```

Figure 20: A demonstration of the in-context example used for SQL generation (Step 3) on FINQA.