A-STAR: Toward enhancing LLM's content-aware capability for table reasoning

Anonymous ACL submission

Abstract

Current methods for table reasoning with LLMs can be broadly categorized into two distinct approaches: text reasoning and code generation, which leverage natural language processing and programming paradigms, respectively. The former is subject to table's scale for the context length limit of LLMs; the latter incurs structural bias due to the lack of awareness of table data. This paper proposes A-STAR, a table reasoning architecture that enhances LLM's table content-aware capability at any scale. Considering the various distributions of records related to various questions in the original table, a decompose-recombine algorithm is introduced to obtain a refined table by decomposing the original table into sub-tables and recombining the records related to question extracted from them. According to the characteristics of these tables, an adaptive strategy will be adopted to select different solvers to generate multiple candidate answers and assign priorities to them. Finally, a semantic-based voting mechanism is designed to fuse these answers to obtain the final response. The experiment shows that A-STAR has achieved state-of-the-art performance in both table-based fact verification and question answering tasks. Our code is available at https://anonymous.4open.science/r/A-STAR-D9DF/.

1 Introduction

Table serves as a fundamental format for representing structured relational data. While current large language models (LLMs) excel at many text-based tasks without task-specific model structure or training data, relying only on designing input prompts (Brown et al., 2020; Wei et al., 2022; Wang et al., 2022; Zhou et al., 2022; Kojima et al., 2022; Li et al., 2022), they still face challenges in table reasoning because of the complex structured characteristics of tabular data. The existing LLM-based table reasoning works can be systematically classified into two principal paradigms: text reasoning based on natural language (Chen, 2022; Ye et al., 2023) and code generation based on symbolic reasoning (Cheng et al., 2022; Zhang et al., 2023; Wang et al., 2024b; Nahid and Rafiei, 2024), employing linguistic processing and computational logic respectively.

In code generation, LLM is prompted to generate code based on question and table schema to operate on the table, utilizing its inherent coding capability to adopt symbolic reasoning, such as SQL statements (Nahid and Rafiei, 2024) or python programs (Wang et al., 2024b). Although this approach achieves scale-agnostic adaptability because it does not need to access table contents, it relies heavily on LLM's inherent coding pattern to infer the relevant records from question and table schema, which may introduce structural bias into the generated code (Jiang et al., 2024; Wang et al., 2024a). Consequently, the code may not accurately reflect the specific content of the table, leading to incorrect responses. In text reasoning, LLM is prompted to reason with question and table data using natural language, benefiting from LLM's semantic parsing capability constructed from training. (Ye et al., 2023) prompts LLM to return a subset of the given table containing only relevant records, and then answer the question based on this subset. Although this approach allows LLM to be aware of all table contents related to the question, thus improving the reliability of reasoning, the performance is limited by the context length of LLM, which restricts the application for larger tables. Through many experiments, (Liu et al., 2023) compared these two approaches, text reasoning performs better on small-scale tables, while code generation dominates at large-scale tables.

In summary, the limitations of existing table reasoning methods force us to explore how a model can be aware of table content at any scale. We pro-

pose A-STAR, an Adaptive Strategy with TAble Refinement for table reasoning. Considering the various distributions of records related to various questions in the original table, a decompose-recombine algorithm is designed to decompose the original table into several sub-tables, and recombine the relevant records extracted from them into a refined table, and an adaptive strategy is raised to generate candidate answers with LLM from sub-tables and refined table and assign different priorities to these answers. To reduce the inherent inconsistency of the response form generated by LLM, a semantic-based voting is conducted on all candidate answers. We evaluate A-STAR using different LLMs as the backbone model on table reasoning tasks involving table-based question answering and fact verification. The experiment results indicate that A-STAR outperforms all existing LLM-based baselines.

The major contributions of this paper are listed below:

- We propose an architecture for table reasoning, namely A-STAR, that aims to enhance LLM's table content-aware capability at any scale.
- An algorithm based on decompose-recombine is introduced to obtain a refined table which LLM can handle effectively by decomposing the original table into sub-tables and recombining the relevant records extracted from them. Additionally, a process including adaptive solving and priority assigning is designed to get multiple candidate answers according to specific table types, they are then fused based on semantics to obtain the final response.
- Our method has achieved leading performance compared to other LLM-based methods in table-based question answering and fact verification tasks. We further discussed the significant superiority of our method through more experiments.

2 Related Works

Table Reasoning with Pre-trained Model Researchers have proposed some pre-trained language models on table reasoning (Herzig et al., 2020; Liu et al., 2021; Zhao et al., 2022; Yin et al., 2020; Gu et al., 2022). They have rich knowledge learned from large-scale data by collecting or synthesizing them. However, these methods require a large amount of corpus related to the task, which results in a high expense. In addition, they may also lead to overfitting problems, reducing their generalization.

Table Reasoning with LLM Compared with pre-trained models, LLM performs well on various tasks with zero-shot prompts, demonstrating the strong generalization. Therefore, there have been many LLM-based table reasoning methods recently. The paradigm includes two types: code generation and text reasoning. For code generation, Binder (Cheng et al., 2022) prompts LLM to identify and solve the parts of the question which cannot be solved by the original program such as schema linking. TabSQLify (Nahid and Rafiei, 2024) prompts LLM to decompose table by SQL statements. ReAcTable (Zhang et al., 2023) introduces ReAct (Yao et al., 2022) into table reasoning. It prompts LLM using external tools to operate table, generating intermediate tables to gradually enhance data, thereby converting it into a more accessible format for an easier solution. Similarly, Chain-of-Table (Wang et al., 2024b) introduce CoT (Wei et al., 2022). The difference is that it defines a set of table operations, and then prompts LLM to generate a table reasoning chain that calls those operations. Compared with the above methods, Chain-of-Table partly reduces the bias caused by letting LLM generate codes directly. For text reasoning, Dater (Ye et al., 2023) prompts LLM to decompose table and question to solve complex tasks. Besides, H-STAR (Abhyankar et al., 2024) discussed the issue of integrating text reasoning and code generation methods. However, it just simply prompts LLM to choose what strategy to use for analysis, without delving into the strengths and the limitations of different strategies, thus lacking interpretability.

3 Methodology

3.1 Overview

We propose A-STAR, an architecture for LLM-based table reasoning tasks. It is scale-free and can fully unleash the potential of LLM by enhancing its table content-aware capability. As illustrated in Figure 1, A-STAR includes a table refiner that decomposes the original table into sub-tables and then recombines relevant records obtained by the extraction of sub-tables into a refined table; an adaptive solver that can adopt different strategies to obtain multiple candidate answers; and an output summarizer for output processing, which can summarize

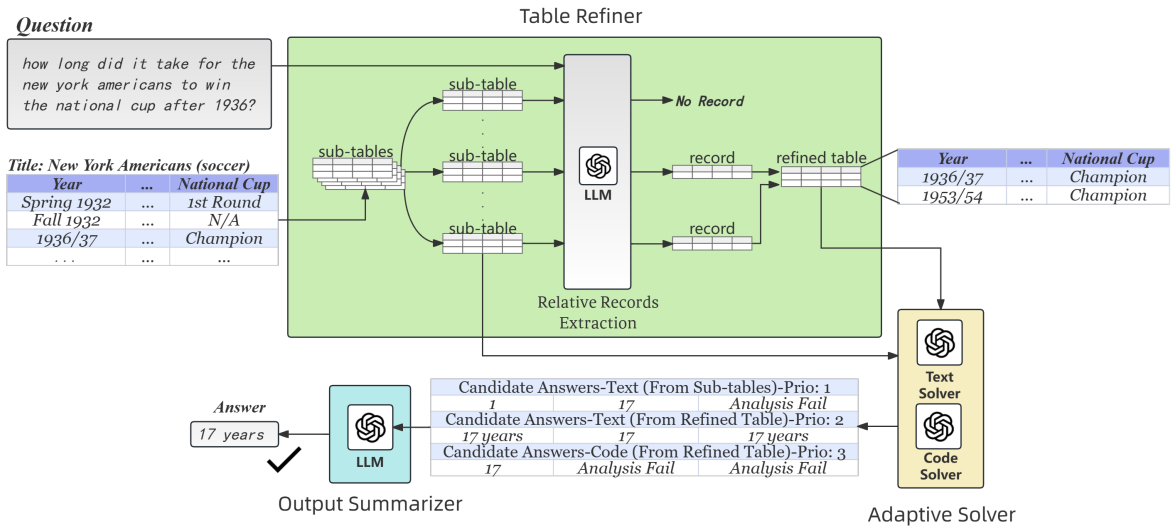

Figure 1: Overview of A-STAR, which consists of three parts: (i) Table Refiner refines the original table into a refined table which has a higher density of relevant records; (ii) Adaptive Solver uses an adaptive strategy on different tables to generate multiple candidate answers and assign different priorities to them and (iii) Output Summarizer unifies semantically identical answers and vote to select the final answer.

all candidate answers and vote based on semantics to generate the final response.

3.2 Table Refiner

How to enhance the table content-aware capability of the model is the key for table reasoning task. Unlike other table-based tasks, table reasoning task requires the model to have a more specific understanding of the table down to the record level. Therefore, accurately locating relevant records is crucial. (Ye et al., 2023) used text reasoning to accomplish this task, prompting LLM to find all relevant records in the given table. However, it's difficult to search the entire table at once due to the context length limit of LLMs. Code generation method (Zhang et al., 2023; Wang et al., 2024b) prompts LLM to combine keywords from question and table schema to infer the relevant records, while the model does not actually understand the table data. Therefore, when the question is ambiguous, code generation raises errors.

We design an algorithm based on decompose-recombine to deal with it. The original table will be split into several sub-tables and a two-step sampling process for them is used to get the refined table. Algorithm 1 describes the overall process.

Table Split Line 1 of Algorithm 1 describes this step. The scale of each sub-table should be considered to ensure that LLM can get enough information at once while avoiding long context lengths impairing the performance. Firstly, the original table will be equally divided into three parts. And then, if their scales are too large or too small, they will be re-divided according to a pre-set length. Finally, the remaining records are merged into an extra sub-table. In this way, the original table has been split into a batch of sub-tables that are appropriately sized and sufficiently uniform.

Algorithm 1 Table Refinement Algorithm

Require: Table T, Question Q
Ensure: Table T'

```
1:  subs = split_into_subtables(T)
2:  Dive the subs into 3 clusters.
3:  for mid in each clusters do
4:      prompt = gen_prompt(mid, Q)
5:      records.append(LLM(prompt))
6:      if records not null then
7:          for each adj to the mid do
8:              prompt = gen_prompt(adj, Q)
9:              records.append(LLM(prompt))
10:         end for
11:     end if
12: end for
13: T' = records
```

Two-Step Sampling Process Lines 2 to 9 of Algorithm 1 describe this step. It's not wise to search all of these sub-tables considering time and expense. For a question, the distribution of its

relevant records in the table contains three situations: (1) Dense. Like "*What is the title of the next episode of a certain episode?*", whose relevant records exist adjacently; (2) Moderate. Like "*How long did it take for a certain team to win the championship again after a certain year?*", whose relevant records exist in non-adjacent sub-tables; (3) Wide. Like "*How many types of competitions did a certain athlete participate in?*", which do not clearly mention the specific data, so the relevant records can exist in all the sub-tables. Based on the above analysis, we propose a two-step sampling process. **Step 1**: Categorize all sub-tables into three clusters, and select the middle sub-table for each cluster. Lines 2 to 5 of Algorithm 1 describe this step. **Step 2**: Select two adjacent sub-tables from the same cluster if there are any relevant records in the selected sub-table. Lines 6 to 9 of Algorithm 1 describe this step.

Relevant Records Extraction LLM is prompted to judge whether the sub-table contains records related to the question, and if so, return their row numbers. After the extraction process is completed, all these rows are merged into a refined table for subsequent analysis.

Algorithm 2 Adaptive Solving Algorithm

Require: Original Table T, New Table T', Set of Sub-tables T_s, Question Q, Minimum Length of Table l_{min}, Number of analysis n
Ensure: Set of Candidate Answers A

1: **if** length of $T < l_{min}$ **then**
2: $text\_prompt$ = gen_text_prompt(T, Q)
3: A.append(LLM($text\_prompt$, n))
4: $code\_prompt$ = gen_code_prompt(T, Q)
5: A.append(LLM($code\_prompt$, n))
6: assign_priority(A, type_a)
7: **else**
8: **for** each sub in T_s **do**
9: $text\_prompt$ = gen_text_prompt(sub, Q)
10: A.append(LLM($text\_prompt$, n))
11: **end for**
12: $text\_prompt$ = gen_text_prompt(T', Q)
13: A.append(LLM($text\_prompt$, n))
14: $code\_prompt$ = gen_code_prompt(T', Q)
15: A.append(LLM($code\_prompt$, n))
16: assign_priority(A, type_b)
17: **end if**

3.3 Adaptive Solver

(Liu et al., 2023) compared the performance of using text reasoning or code generation to solve table reasoning tasks through many experiments. The results show that the accuracy of text reasoning performs better when the model can access the complete table. LLM is mainly trained on natural language data, which means its capability of code generation is inherently lower than text reasoning. However, large-scale tables in real-world scenarios limits the potential of LLM's text reasoning capability. Conversely, an obvious advantage of code generation is that it can answer a question under the circumstances of accessing only the schema of the table, allowing it to handle situations with a large table scale. However this advantage cannot fully cover the additional bias introduced by code generation methods due to the lack of awareness of table content. In summary, both these methods have their own strengths and limitations. We must carefully design the strategy that is flexible enough to face different situations.

An adaptive strategy is introduced to solve this problem. It can flexibly select different solvers to adapt on a sub-table or refined table to get multiple candidate answers and assign priorities to these answers. Algorithm 2 describes the overall process.

Solvers Call LLM is prompted to act as different solvers. Text solver directly gives the answer or a "*can't be answered*" judgment, while code solver generates code to run on the original table to obtain the answer or an execution error. To reduce runtime errors in the generated code, the input and the output are defined in advance. To further enhance the diversity, self-consistency (Wang et al., 2022) that repeatedly calls the solvers multiple times is used to generate candidate answers.

Priority Assignment The adaptive strategy consists of three parts: **(1) Check the scale of original table**. The refinement is not necessary when the original table is small. In this case, the candidate answers from text reasoning will be considered more reliable than from code generation, corresponding to 'type_a' in Line 6 of Algorithm 2. **(2) The candidate answers from sub-tables should be given the highest priority**. The sub-tables sampled will not only be used to extract relevant records, but they will also be analyzed using text reasoning. That's because, first, if all relevant records exist in the same sub-table, this process will more likely

obtain the correct answer. Second, since each sub-table contains different records, it's unlikely that all candidate answers will be the same, so they will not cause significant interference to the subsequent analysis. In a word, a simple text reasoning will be conducted on all sub-tables, excluding failed processes and assigning the highest priority to the remaining candidate answers. **(3) Prioritizing codes to analyze the refined table**. Regardless of the scale of the table after refinement, the candidate answers obtained by code generation will be assigned higher priority, corresponding to 'type_b' in Line 16 of Algorithm 2. Text reasoning is only suitable when all relevant records are accessible, while our table refinement algorithm sacrifices accuracy for efficiency and thus cannot guarantee that all relevant records are found. Therefore, the strategy tends to code generation. Moreover, previous LLM-based coding-dependent methods simply extracted the first few records of the table as the view provided to LLM. In contrast, our method provides LLM with a view that covers more relevant records, thereby improving the coding performance. In summary, all candidate answers are assigned different priorities through this step, which will affect the subsequent voting process.

3.4 Output Summarizer

LLM prefers to add unnecessary context or explanation in response, leading to the inconsistency of the form. For example, for the question "*How old is Bob's father?*", from a semantic perspective, "*48*", "*48 years old*", "*Bob's father is 48 years old this year*" express the same meaning. But from a string comparison perspective, only one of them can be considered correct. Previous works mitigated this impact by designing complex string processing, while our method utilizes LLM to accomplish this task for its clear advantage in performing semantic-related tasks. Overall, in each process LLM is prompted to compare two candidate answers selected, and give a judgment on whether they are semantically identical. If yes, they will be unified into the same answer.

The answer with the highest number of appearances among the candidate answers will be selected through voting. In this process, priority will only be considered when some answers appear the same number of times. It means the answer that appears more frequently is still preferred despite its low priority.

4 Experiments

4.1 Experimental Setup

Datasets We evaluate A-STAR on three commonly-used table reasoning datasets: TabFact (Chen et al., 2019), WikiTQ (Pasupat and Liang, 2015) and FeTaQA (Nan et al., 2022). TabFact is a table-based fact verification dataset, whose task is to judge whether a statement is true or not based on table data. We report results on its small test set. WikiTQ is a table-based question answering dataset that focuses on answering questions through complex inference of the table data. FeTaQA is a more difficult table-based question answering dataset as it requires longer, freer responses.

Baselines We report the LLM-based methods that have performed well on the above datasets, including Binder (Cheng et al., 2022), Dater (Ye et al., 2023), ReAcTable (Zhang et al., 2023), TabSQLify (Nahid and Rafiei, 2024), Chain-of-Table (Wang et al., 2024b) and H-STAR (Abhyankar et al., 2024). We also report some basic methods, including (a) Text End-to-End, which prompts LLM to directly give the answer, (b) Text CoT, which prompts LLM to think and give a reasoning chain before answering, (c) Code End-to-End, which prompts LLM to write code and (d) Code CoT, which prompts LLM to think and give a reasoning chain before writing code. Moreover, we also report some pre-trained methods including TaPas (Herzig et al., 2020), Tapex (Liu et al., 2021), ReasTAP (Zhao et al., 2022), OmniTab (Jiang et al., 2022), LEVER (Ni et al., 2023) and PASTA (Gu et al., 2022).

In order to discuss the impact of different backbone models of A-STAR, we evaluate it based on Llama3.1-8b (Dubey et al., 2024), Qwen2.5-7b (Yang et al., 2024) and GPT-4o-mini (Achiam et al., 2023). We reran the tests using codes provided by their authors. For those who have not open-sourced the prompt and the code on some datasets, considering the impact of code (especially the design of their prompt templates) on the performance, we have decided not to report the performance of these methods on the corresponding dataset.

Metrics TabFact and WikiTQ both use Execution Accuracy (EA) as the evaluation metric, we follow this practice. For TabFact, we prompt LLM to only generate "*true*" or "*false*" and then evaluate

Methods	Llama3.1-8b		Qwen2.5-7b		GPT-4o-mini	
	TabFact	WikiTQ	TabFact	WikiTQ	TabFact	WikiTQ
Text End-to-End	55.09	22.44	58.15	20.42	62.01	30.62
Text CoT	60.38	26.52	63.24	23.07	67.34	36.9
Code End-to-End	30.63	24.7	44.91	35.7	60.08	46.94
Code CoT	41.85	29.58	47.97	39.8	64.28	49.75
Binder	51.98	27.14	53.51	29.7	67.05	38.03
Dater	57.9	37.41	58.99	40.01	74.7	52
ReAcTable	-	37.64	-	40.42	-	57.09
TabSQLify	60.67	44.96	59.39	47.38	78.26	68.76
Chain-of-Table	63.83	-	62.35	-	85.13	-
H-STAR	70.1	51.06	64.38	51.24	89.38	74.95
A-STAR	**75.69**	**58.49**	**73.22**	**59.46**	**89.82**	**78.5**

Table 1: Performance of various methods on TabFact and WikiTQ datasets. '-' indicates that we don't report the result because of the irreproducibility.

the answer using string matching. For WikiTQ, we directly compare the model answer with the gold answer. We don't use the Python-based WikiTQ evaluator (Pasupat and Liang, 2015), as our output summarizer takes this responsibility. Similarly, we evaluate results on FeTaQA by calculating ROUGE (Lin, 2004) score of the model answer and the gold answer.

4.2 Experiment Results

Table 1 shows some of the results. Due to space limitations, we have the additional results in Appendix A, including comparisons with pre-trained methods and evaluation on the FeTaQA dataset. In summary, A-STAR outperforms all existing LLM-based methods with different datasets and backbone models. It is worth mentioning that previous methods sampled from the train set to construct a few-shot prompt, while we do not use any examples, which reduces the cost of generalizing our method to other table reasoning tasks and leaves room for further improvement.

We notice that A-STAR shows a decrease in improvement compared to other methods when using a stronger LLM as the backbone model. That's because, with a stronger LLM, the length of context it can handle increases, which enhances the performance of text reasoning; and also its coding skill improves, which makes the code generation process perform better. In contrast, A-STAR benefits from the performance improvement of the LLM itself only when calling it to extract relevant records from sub-tables. Besides, when generating an answer, the performance of coding-dependent methods will be greatly affected by the differences in LLMs' coding capabilities, while it has limited impact on the performance of A-STAR as our adaptive solving algorithm. In other words, we aim to maximize the potential of weak LLM, so the improvement of the LLM itself will to some extent compensate for the gap in our performance compared to other methods.

4.3 Ablation Study

To evaluate the effectiveness of each module in A-STAR, we conduct an ablation study on the WikiTQ dataset. We remove the table refiner, instead we always select the first sub-table. We remove the adaptive solver and only use the text solver or the code solver to generate candidate answers. Moreover, we shuffle the candidate answers to eliminate the influence of the priority. We remove the output summarizer, instead choosing the first candidate answer as the result.

Table 2 shows the specific contribution of each module to the A-STAR's performance. Overall, removing any module will result in a performance loss, emphasizing the importance of all modules in A-STAR. Specifically: (a) Removing the table refiner has the least impact on performance, for there are a considerable number of questions whose answers can be directly obtained from the first few records in the table. Therefore, our alternative, which chooses the first sub-table has little impact on answering these questions. (b) After removing the adaptive solver, the alternative of using only

	Llama3.1-8b	GPT-4o-mini
A-STAR	**58.49**	**78.5**
w/o Table Refiner	49.06	71.59
w/o Adaptive Solver(ALL Text Solver)	48.73	68.76
w/o Adaptive Solver(ALL Code Solver)	44.66	67.4
w/o Output Summarizer	44.34	56.4

Table 2: Performance of A-STAR removing (i) Table Refiner, (ii) Adaptive Solver and (iii) Output Summarizer on WikiTQ dataset.

the text solvers performed better than using only the code solvers. That's because text solvers can keep a stable performance on a small table, while code solvers are likely to generate code with errors on any table. It can be seen that when using only code solvers, the decrease in performance using Llama3.1-8b as the backbone model is more significant compared to using GPT-4o-mini, which reflects that the difference in LLM's inherent coding capability will significantly affect the performance of coding-dependent methods. (c) Removing the output summarizer has the greatest impact on performance. Due to our priority assignment strategy assigning the highest priority to the candidate answers obtained from sub-tables, the alternative will always choose them. However, without priority, the accuracy of the candidate answers obtained from a sub-table cannot be guaranteed as they may not necessarily contain relevant records.

Methods	Table Scale(Num. of Tokens)		
	<2k	2k-4k	>4k
Binder	33.59	26.65	5.13
Dater	48.65	28.53	13.68
ReAcTable	46.33	30.09	20.51
TabSQLify	52.7	37.93	30.77
H-STAR	56.95	45.45	40.17
A-STAR	**65.25**	**52.04**	**46.15**

Table 3: Performance of various methods on different scales of tables from WikiTQ dataset, using Llama3.1-8b as backbone model.

4.4 Discussion

(Chen, 2022) points out that when the number of table rows exceeds 30, the performance bottleneck that a LLM can handle has been reached. To evaluate the impact of the table scale, we not only consider the number of rows but also the lengths. Specifically, we classify tables in the WikiTQ dataset into three parts by the number of tokens: tables with less than 2000 tokens, tables with more than 4000 tokens and tables with tokens from 2000 to 4000.

The result is in Table 3. All methods show a decrease in performance when the table scale increases. Relatively, A-STAR controls the magnitude, thus ensuring a consistent performance as much as possible. This is attributed to our extraction algorithm based on decompose-recombine and the adaptive priority assignment strategy successfully reducing the impact of table scale on the performance.

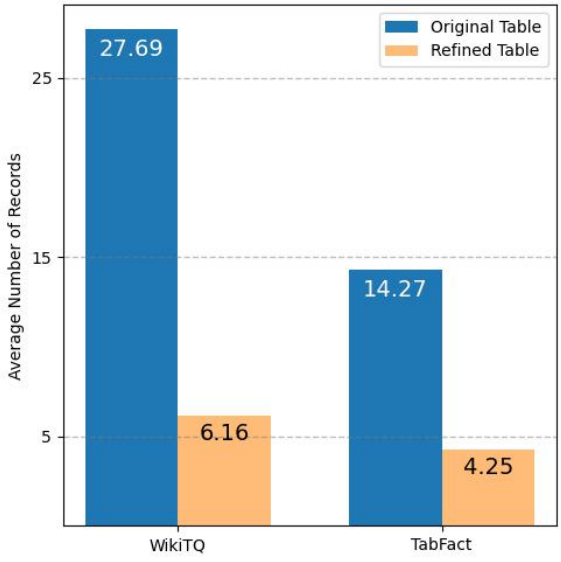

Figure 2: Comparison of the scale of original table/refined table. We exclude all the small tables as they do not require refinement.

Table refiner refines the original table into a table containing more relevant records and less redundant records through the algorithm based on decompose-recombine. Figure 2 compares the average scale of the original table and the refined table. We exclude all small tables that do not require refinement. It can be seen that table refiner

significantly reduces the number of records in the refined table - from an average of 27.69 records to 6.16 records for the WikiTQ dataset, and from an average of 14.27 records to 4.25 records for the TabFact dataset. It demonstrates how our table refiner effectively reduces the redundant information unrelated to the question and allows the LLM to focus more on valuable information for inference.

	Llama3.1-8b	GPT-4o-mini
A-STAR	**58.49**	**78.5**
w/o SC	45.49	67.29

Table 4: Performance of A-STAR with/without Self-Consistency on WikiTQ dataset.

Table 4 shows the impact of with/without self-consistency on the performance of A-STAR. We use self-consistency only when analyzing the refined table but not the sub-tables, so the weight of candidate answers obtained from sub-tables has been weakened. In other words, our strategy ensures that this part of candidate answers have the highest priority while reducing their negative impact from the randomness of these answers to the result. It's worth noticing that after using a stronger LLM as the backbone model, the performance loss without self-consistency decreased from 22.23% to 14.28%. It can be explained by the fact that as the capability of the LLM improves, it can obtain the correct answer using any analysis method.

	Llama3.1-8b	GPT-4o-mini
in Text Solver	23.27%	13.42%
in Code Solver	18.34%	11.95%
in Both	**31.66%**	**61.32%**
in Neither	26.73%	13.31%

Table 5: The proportion of correct answer in text solver/code solver/both of them/neither of them on the WikiTQ dataset.

We calculate the proportion of correct answers appearing only in the text solver/only in the code solver/in both/in neither. Table 5 shows the result. It can be seen that (a) the most common situation is that both the text solver and the code solver can obtain the correct answer, accounting for 31.66% and 61.32%. It indicates that there is actually no essential difference between these two methods, but how we make choices based on the actual situations is important. In addition, after using a stronger LLM, the proportion of this situation has reached an absolute advantage, which once again proves the viewpoint that as the capability of the LLM improves, it can obtain the correct answer using any analysis method. (b) The correct answer appears more frequently in the text solvers than in the code solvers. There are two reasons for this. First, text solvers can keep a stable performance on a small table, while code solvers are likely to generate code with errors on any table. Second, when analyzing sub-tables, we only use the text solver. In some cases, all relevant records may exist in the same sub-table, so the text solver can directly obtain the correct answer, which cannot be achieved by code solvers.

	Llama3.1-8b	GPT-4o-mini
A-STAR	**58.49**	**78.5**
w/o Unification	38.26	67.52

Table 6: Performance of A-STAR with/without Semantic-Consistency on WikiTQ dataset.

Table 6 shows the impact of with/without semantic unification on the performance of A-STAR. After removing this step, there was an astonishing and significant decline in the result. We emphasize that it DOES NOT demonstrate the importance of unification itself. If there were no such process, the candidate answers will only be voted on with simple string matching, thus leading to our adaptive solving method losing its expected consistency in the result. In other words, the performance degradation without unification is actually a continuation of the impact of our adaptive solving method on A-STAR's performance, which proves the importance of this module from the side.

5 Conclusion

In this paper we propose A-STAR, an architecture for table reasoning that aims to enhance LLM's content-aware capability at any scale. It includes a decompose-recombine algorithm to refine the original table; an adaptive strategy to obtain various answers and assign priorities to them; and a semantic-based voting process to generate the final response. Experiment results have shown that A-STAR outperforms all existing LLM-based methods in typical table reasoning tasks including question answering and fact verification. Further analysis highlights the significance of our method.

Limitations

Our method has not been experimented on more table reasoning tasks such as Text-to-SQL, table-based prediction and tabular data analysis, etc., which imposes a limit on the generalization. In addition, the datasets used are constructed based on Wikipedia, the tabular data is simply organized, clear, easy to understand and not too large. However, tables in real-world scenarios raise more challenges including extremely large scale, noisy records and complex organization forms. How to address these challenges remains an unresolved issue.

Moreover, our method further increases the complexity compared to previous LLM-based methods, involving repeated calls and interactions between multiple LLMs, which may lead to a possible decrease in efficiency. How to find a trade-off between performance and efficiency is the topic we will study next.

References

Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K. Reddy. 2024. H-star: Llm-driven hybrid sql-text adaptive reasoning on tables. *ArXiv*, abs/2407.05952.

OpenAI Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, Jeff Belgum, Irwan Bello, and 260 others. 2023. Gpt-4 technical report.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Wenhu Chen. 2022. Large language models are few(1)-shot table reasoners. *ArXiv*, abs/2210.06710.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact: A large-scale dataset for table-based fact verification. *arXiv preprint arXiv:1909.02164*.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir R. Radev, Marilyn Ostendorf, Luke S. Zettlemoyer, Noah A. Smith, and Tao Yu. 2022. Binding language models in symbolic languages. *ArXiv*, abs/2210.02875.

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, and 510 others. 2024. The llama 3 herd of models. *ArXiv*, abs/2407.21783.

Zihui Gu, Ju Fan, Nan Tang, Preslav Nakov, Xiaoman Zhao, and Xiaoyong Du. 2022. Pasta: table-operations aware fact verification via sentence-table cloze pre-training. *arXiv preprint arXiv:2211.02816*.

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *ArXiv*, abs/2406.00515.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. Omnitab: Pretraining with natural and synthetic data for few-shot table-based question answering. *arXiv preprint arXiv:2207.03637*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *ArXiv*, abs/2205.11916.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, B. Chen, Jian-Guang Lou, and Weizhu Chen. 2022. Making language models better reasoners with step-aware verifier. In *Annual Meeting of the Association for Computational Linguistics*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Annual Meeting of the Association for Computational Linguistics*.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. Tapex: Table pre-training via learning a neural sql executor. *arXiv preprint arXiv:2107.07653*.

Tianyang Liu, Fei Wang, and Muhao Chen. 2023. Rethinking tabular data understanding with large language models. In *North American Chapter of the Association for Computational Linguistics*.

Md Mahadi Hasan Nahid and Davood Rafiei. 2024. Tabsqlify: Enhancing reasoning capabilities of llms through table decomposition. *ArXiv*, abs/2404.10150.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, and 1 others. 2022. Fetaqa: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

Ansong Ni, Srini Iyer, Dragomir R. Radev, Ves Stoyanov, Wen tau Yih, Sida I. Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. *ArXiv*, abs/2302.08468.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Zhijie Wang, Zijie Zhou, Da Song, Yuheng Huang, Shengmai Chen, Lei Ma, and Tianyi Zhang. 2024a. Where do large language models fail when generating code? *arXiv preprint arXiv:2406.08731*.

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. 2024b. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *ArXiv*, abs/2401.04398.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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, and 25 others. 2024. Qwen2.5 technical report. *ArXiv*, abs/2412.15115.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2023. Reactable: Enhancing react for table question answering. *Proc. VLDB Endow.*, 17:1981–1994.

Yilun Zhao, Linyong Nan, Zhenting Qi, Rui Zhang, and Dragomir Radev. 2022. Reastap: Injecting table reasoning skills during pre-training via synthetic reasoning examples. *arXiv preprint arXiv:2210.12374*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and 1 others. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

A Supplementary Experiments

Methods	TabFact	WikiTQ
TaPas	83.9	48.8
Tapex	85.9	57.5
ReasTAP	86.2	58.6
OmniTab	-	62.8
LEVER	-	65.8
PASTA	**90.6**	-
A-STAR with Llama3.1-8b	75.69	58.49
A-STAR with Qwen2.5-7b	73.22	59.46
A-STAR with GPT-4o-mini	89.82	**78.5**

Table 7: Performance of pre-trained methods and A-STAR with different backbone models on TabFact and WikiTQ datasets.

A.1 Comparison with Pre-trained Methods

Table 7 shows the performance of pre-trained methods and A-STAR with different backbone model on TabFact and WikiTQ datasets. We do not rerun the test but faithfully report the results from the original paper.

A.2 Evaluation on FeTaQA dataset

Table 8 shows the result of some LLM-based methods on the FeTaQA dataset with different backbone models. Unlike other table reasoning tasks, FeTaQA requires the model to provide a descriptive statement as the answer rather than directly giving the result. And for evaluation, FeTaQA calculates the matching degree between the model statement and the gold statement to assess the quality of the answer. To address this requirement, we have slightly modified the prompts to make the LLM tend to generate longer and more detailed answers.

It's worth mentioning that using only simple text reasoning + CoT can achieve a performance that is close to or even better than LLM-based methods.

Methods	Llama3.1-8b			Qwen2.5-7b			GPT-4o-mini		
	1	2	L	1	2	L	1	2	L
Text End-to-End	0.42	0.21	0.37	0.44	0.22	0.38	0.56	0.34	0.48
Text CoT	**0.48**	0.24	0.42	0.46	0.24	0.41	0.58	0.35	**0.52**
Code End-to-End	0.27	0.13	0.25	0.29	0.13	0.25	0.36	0.18	0.33
Code CoT	0.28	0.13	0.25	0.31	0.15	0.28	0.38	0.19	0.34
TabSQLify	0.41	0.2	0.37	0.37	0.16	0.33	0.49	0.35	0.44
H-STAR	0.45	0.23	0.4	0.46	0.25	0.43	0.54	0.44	0.5
A-STAR	**0.48**	**0.25**	**0.45**	**0.49**	**0.27**	**0.44**	**0.63**	**0.46**	**0.52**

Table 8: Performance of various methods on FeTaQA dataset. '1' and '2' means calculating the score using ROUGE-n where n = 1 or 2. 'L' means calculating the score using ROUGE-L.

We attribute it to the evaluation rule of the FeTaQA. ROUGE score measures the completeness of the information related to the gold answer in the model answer. In fact, we observed that when the model generates a longer answer, it's easier to obtain a higher ROUGE score, regardless of whether the answer is correct. In addition, LLM excels in using descriptive statements to answer a question, which allows it to generate a response that has a similar style to the gold answer through a simple prompt.