Rethinking Tabular Data Understanding with Large Language Models

Tianyang Liu
UC San Diego
til040@ucsd.edu

Fei Wang
USC
fwang598@usc.edu

Muhao Chen
UC Davis
muhchen@ucdavis.edu

Abstract

Large Language Models (LLMs) have shown to be capable of various tasks, yet their capability in interpreting and reasoning over tabular data remains an underexplored area. In this context, this study investigates from three core perspectives: the robustness of LLMs to structural perturbations in tables, the comparative analysis of textual and symbolic reasoning on tables, and the potential of boosting model performance through the aggregation of multiple reasoning pathways. We discover that structural variance of tables presenting the same content reveals a notable performance decline, particularly in symbolic reasoning tasks. This prompts the proposal of a method for table structure normalization. Moreover, textual reasoning slightly edges out symbolic reasoning, and a detailed error analysis reveals that each exhibits different strengths depending on the specific tasks. Notably, the aggregation of textual and symbolic reasoning pathways, bolstered by a mix self-consistency mechanism, resulted in achieving SOTA performance, with an accuracy of 73.6% on WikiTableQuestions, representing a substantial advancement over previous existing table processing paradigms of LLMs.[1]

1 Introduction

Large Language Models (LLMs; Brown et al. 2020; Chowdhery et al. 2022; Zhang et al. 2022; OpenAI 2022, 2023a,c; Touvron et al. 2023a,b; Li et al. 2023b; Lozhkov et al. 2024) have revolutionized the field of NLP, demonstrating an extraordinary ability to understand and reason over rich textual data (Wei et al., 2023; Wang et al., 2023; Zhou et al., 2023; Kojima et al., 2023; Li et al., 2023c). On top of LLMs' existing capabilities for NLP, further bolstering their potential for decision-making by drawing from external knowledge sources remains an exciting research frontier (Nakano et al., 2022; Mialon et al., 2023; Hao et al., 2023; Jiang et al., 2023b). Amongst such knowledge sources, tabular data serve as a ubiquitous kind due to their expressiveness for relations, properties and statistics, and their being easy to construct by human curators (Chen et al., 2020; Wang et al., 2021; Xie et al., 2022).

Like humans, LLMs can also benefit from reading tabular data accompanying text. However, as indicated in Fig. 1, the structural nature of tables

Marek Plawgo

Year	1999	2000	...	2008	2012
Competition	European Junior Championships	World Junior Championships	...	Olympic Games	European Championships
Venue	Riga, Latvia	Santiago, Chile	...	Beijing, China	Helsinki, Finland
Position	4th	1st	...	7th	18th (sf)
Event	400 m hurdles	400 m hurdles	...	4x400 m relay	400 m hurdles
Notes	52.17	49.23	...	3:00.32	50.77

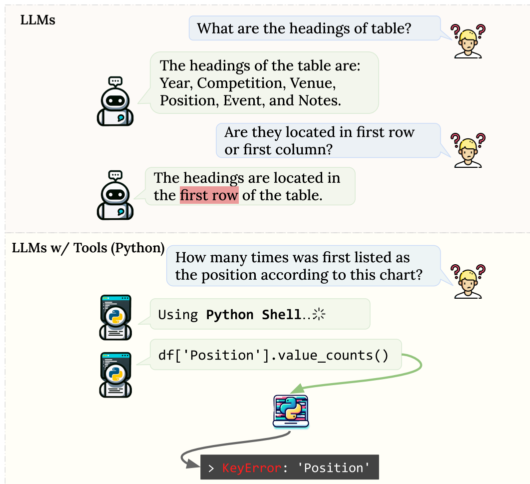

Figure 1: Demonstration of the challenges faced by LLMs in comprehending and interpreting table structures. In the first example, the LLM correctly identifies table headings but struggles to accurately determine their positions within the table structure. In the second example, the model using Python Shell as an external tool incorrectly interprets headings (located in the first column) as column headers, leading to subsequent errors in the generated code. Some logos in this and subsequent figures are generated using OpenAI's DALL-E3 (OpenAI, 2023b).

[1]Our code is available at https://github.com/Leolty/tablellm.

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 450–482
June 16-21, 2024

World War II casualties of Poland

Description Losses	Direct War Losses	Murdered	Deaths In Prisons & Camps	Deaths Outside of Prisons & Camps	Murdered in Eastern Regions	Deaths other countries	Total
1939/40	360,000	75,000	69,000	-	-	-	504,000
1940/41	-	100,000	210,000	42,000	-	-	352,000
...	...	...	...	...	...	-	...
1943/44	-	82,000	381,000	218,000	-	-	681,000
1944/45	183,000	-	-	-	100,000	-	270,000
Total	...	506,000	1,146,000	473,000	100,000	2,000	2,770,000

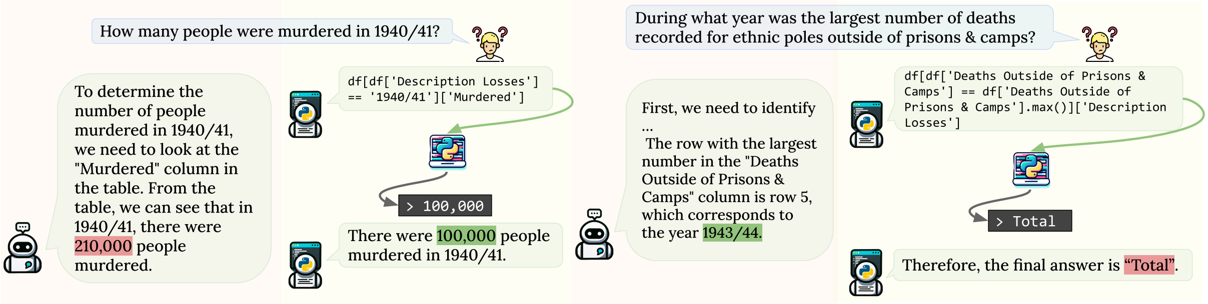

Figure 2: Illustrative examples sampled from the WikiTableQuestions dataset, comparing textual reasoning (via direct prompting) and symbolic reasoning (via python shell interactions). ***Top***: The table and its title. ***Bottom Left***: In the first question example, textual reasoning yields an incorrect interpretation due to limitations in precision localization, while symbolic reasoning accurately locates the answer using Python code. ***Bottom Right***: In the second question example, textual reasoning successfully identifies the answer, but symbolic reasoning incorrectly treats the special `total` row as the final answer.

presents unique challenges to these models. Inherently designed to parse and process vast expanses of unstructured textual content, LLMs confront a paradigm shift when facing tabular data. Linearizing tables to suit the LLM paradigm can obscure the inherent structural and relational information, making tasks such as precise localization and complex statistical analyses. Additionally, the design variations in tables, whether 'column tables' with headers in the first row or 'row tables' with headers in the first column, further complicate the interpretation process. Beyond structural concerns, numerical reasoning and aggregation over tabular data present another layer of complexity. While LLMs excel at textual understanding, they occasionally stumble when confronted with tasks necessitating precise numerical computation within tables. Moreover, tables often present a dense amalgamation of textual or numerical data. The sheer volume and intricacy of this information can risk overshadowing crucial details, potentially impeding the LLM's decision-making abilities (Shi et al., 2023).

With the emergence of instruction fine-tuning techniques (Wei et al., 2022; Chung et al., 2022) and the application of Reinforcement Learning from Human Feedback (RLHF) (Stiennon et al., 2022; Gao et al., 2022; Christiano et al., 2017), LLMs have witnessed significant enhancements in their alignment capabilities, paving the way for transitioning from few-shot to zero-shot learning settings (Kojima et al., 2023). In light of these advancements, this paper delves deep into the the challenges and intricacies of tabular understanding and reasoning by LLMs, exemplified in Fig. 2. We organize our exploration around three **pivotal research questions**: (1) How well do LLMs perceive table structures and how can we ensure robustness against structural variations? (2) Comparing textual and symbolic reasoning for table data in LLMs, which prevails in effectiveness, and what advantages and challenges manifest in each strategy? (3) Will the aggregation of multiple reasoning pathways enhance the accuracy and reliability of tabular data interpretation by LLMs?

In pursuit of answering the aforementioned research questions, we conduct experiments on SOTA LLMs such as GPT-3.5 (OpenAI, 2023a). Our findings in §4 underscore that while LLMs are adept at semantically interpreting tables, their capability to resist structural variance (§4.1) and understand table structures (§4.2) is suboptimal. Motivated by these findings, we propose a table structure normalization method to enhance LLMs' resilience against structural table variations in §4.3. Intriguingly, §5.1 reveals that textual reasoning surpasses symbolic reasoning in contexts with limited table content, defying conventional conceptions of symbolic reasoning's dominance in other domains (Mialon et al., 2023). Both textual and symbolic reasoning strategies encompass different advantages and challenges, which is detailed in §5.2. To harness the unique strengths of each, we implement mix self-consistency mechanism (§6) that remarkably attains SOTA performance on Table QA, exemplifying the synergistic potential when both reasoning strategies are aggregated.

2 Related Work

PLMs for Tabular Data Processing. Tabular reasoning presents unique challenges due to the fusion of free-form natural language questions with structured or semi-structured tabular data, for which PLMs jointly trained on tables and text are developed in the past few years, including TaBERT (Yin et al., 2020), TaPas (Herzig et al., 2020), TAPEX (Liu et al., 2022), ReasTAP (Zhao et al., 2022), and PASTA (Gu et al., 2022). The recent development of TableLlama (Zhang et al., 2023a), an open-source model excelling in a variety of table-based tasks, adds a new dimension to the field. Despite these advancements, recent studies have identified generalization issues under table perturbations (Zhao et al., 2023; Chang et al., 2023), raising concerns regarding the robustness of PLMs. Specific efforts like LETA (Zhao et al., 2023) and LATTICE (Wang et al., 2022) have investigated and mitigated the vulnerabilities related to structural perturbations of tabular data, like row/column shuffling and table transpose, through various techniques, including data augmentation and order-invariant graph attention. However, these approaches require whitebox access to the models, limiting their applicability to SOTA LLMs with only blackbox accessibility, a limitation directly addressed in this work.

Tabular Data Processing with LLMs. Recent advancements in LLMs, notably within few-shot learning, have demonstrated their potential for tabular reasoning. Chen (2023) leveraged the Chain-of-Thought (CoT) technique (Wei et al., 2023) to illustrate LLMs' effectiveness in this domain. Building upon CoT, Cheng et al. (2023) and Ye et al. (2023) introduced frameworks that incorporate symbolic reasoning for improved comprehension, with Ye et al. emphasizing their ability to adeptly decompose both evidence and questions. The advent of aligned models, such as ChatGPT, has enabled zero-shot table reasoning. However, these models often lack sensitivity to table structures, struggling with structural perturbations. StructGPT (Jiang et al., 2023a), while introducing a promising framework for LLMs to efficiently engage with structured data, has its effectiveness limited by not integrating symbolic reasoning, a critical aspect for enhancing the full capabilities of LLMs in tabular reasoning, which is the focal point of this study. Furthermore, while programming-based approaches can mitigate some challenges, they are limited in addressing free-form queries, creating a gap in the landscape. Innovations like AutoGPT (Significant Gravitas, 2023) have sought to address this, spawning the development of tabular agents like LangChain (Chase, 2022), SheetCopilot (Li et al., 2023a), and Data-Copilot (Zhang et al., 2023b). These agents offer solutions unattainable through conventional programming but still require rigorous evaluation in various scenarios. In our study, we delve into addressing these challenges for enhancing LLMs' reasoning capabilities within structural perturbations, hence providing insights that facilitate improved accuracy in the current context.

3 Preliminaries

This section succinctly introduces the foundational aspects of our study over structurally perturbed tabular data. §3.1 formally defines the problem, delineating the critical notations and conceptual frameworks, and §3.2 explicates our experimental setup details, elucidating dataset choice, model utilization, and evaluation strategy.

3.1 Problem Definition

Question answering (QA) over tabular data, commonly known as the TableQA task, is an important challenge in NLP. In this study, we targets TableQA to explore and enhance the proficiency of LLMs, in reasoning over tabular data. Additionally, we probe the robustness and adaptability of these models by introducing structural perturbations to tables.

Let $\mathcal{T}$ represent a table consisting of $\mathcal{R}$ rows and $\mathcal{C}$ columns, and τ represent its title/caption. Each cell in $\mathcal{T}$ is denoted by $\mathcal{T}_{i,j}$, where $i \in [0, \mathcal{R}-1]$ and $j \in [0, \mathcal{C}-1]$. $\mathcal{T}_{0,j}$ are headers. Given a question $\mathcal{Q}$ pertaining to the table, our task is to identify an answer $\mathcal{A}$. This answer is generally a collection of values, denoted as $\{a_1, a_2, \ldots, a_k\}$, where $k \in \mathbb{N}^+$.

Furthermore, to delve deeper into the structural comprehension of LLMs, we introduce structural perturbations, which include:[2]

1. **Transposed Table ($\mathcal{T}^\top$):** A table obtained by converting rows to columns and vice-versa, maintaining the row and column order:

$$\mathcal{T}^\top_{i,j} = \mathcal{T}_{j,i} \quad \forall i \in [0, \mathcal{R}-1], j \in [0, \mathcal{C}-1].$$

2. **Row Shuffled Table ($\mathcal{T}_\Pi$):** A table obtained by randomly shuffling the rows (excluding the headers) with a random permutation function π, while keeping the order of columns unchanged:

$$\mathcal{T}_{\Pi_{i,j}} = \mathcal{T}_{\pi(i),j} \quad \forall i \in [1, \mathcal{R}-1], j \in [0, \mathcal{C}-1]$$

3. **Row Shuffled and Transposed Table ($\mathcal{T}^\top_\Pi$):** A table obtained by first randomly shuffling the rows (excluding headers) and then applying transposition:

$$\mathcal{T}^\top_{\Pi_{i,j}} = \mathcal{T}_{j,\pi(i)} \quad \forall i \in [1, \mathcal{R}-1], j \in [0, \mathcal{C}-1]$$

Defining our research problem more formally: our primary objective is to investigate the function, f, that can appropriately answer the posed question using the provided table. Specifically, this function will take three arguments: the table variant $\mathcal{T}' \in \{\mathcal{T}, \mathcal{T}^\top, \mathcal{T}_\Pi, \mathcal{T}^\top_\Pi\}$, its title τ, and the question $\mathcal{Q}$. It will output an answer $\mathcal{A}$. The entire problem can be formally framed as:

$$f(\mathcal{T}', \tau, \mathcal{Q}) \rightarrow \mathcal{A}, \quad \forall \mathcal{T}' \in \{\mathcal{T}, \mathcal{T}^\top, \mathcal{T}_\Pi, \mathcal{T}^\top_\Pi\}$$

3.2 Experimental Setup

This section details the experimental setup adopted in our study, including the datasets employed, model selection, evaluation metrics, reasoning methods, and other details.

Dataset. We used the WikiTableQuestions (WTQ; Pasupat and Liang 2015) dataset for our experiments. The test set comprises 421 tables. Each table provides up to two question-answer pairs; if a table has fewer than two, only one was chosen, totaling 837 unique data points. With our four table configurations (original and three perturbations), the overall evaluation data points amount to $837 \times 4 = 3,348$.

Models. We employ the GPT-3.5 (OpenAI, 2023a) series for our research. Given that tables usually have extensive data, depending on the prompt length, we dynamically use `gpt-3.5-turbo-0613` and `gpt-3.5-turbo-16k-0613`, with a primary aim to optimize cost when querying the API.

Evaluation Metrics. Following prior works (Jiang et al., 2022; Ni et al., 2023; Cheng et al., 2023; Ye et al., 2023), we employ *Exact Match Accuracy* as the evaluation metric to validate predictions against ground truths, embedding instructions in prompts for consistent and parseable outputs.

Reasoning Methods. Our evaluation hinges on two distinct zero-shot reasoning approaches:

- **Direct Prompting (DP)** is a textual reasoning method that prompts LLMs to answer questions in a zero-shot manner. Rather than directly providing the answer, LLMs are instructed to reason step-by-step before concluding. More details can be found in Appx. §A.1,
- **Python Shell Agent (PyAgent)** is a symbolic reasoning approach where the model dynamically interacts with a Python shell. Specifically, LLMs use the Python Shell as an external tool to execute commands, process data, and scrutinize results, particularly within a pandas dataframe, limited to a maximum of five iterative steps. Detailed prompt is presented in Appx. §A.2.

Other Details. Depending on the scenario, we adjust the temperature setting. In cases not employing self-consistency, we set it to 0. For scenarios involving self-consistency, the temperature is set to 0.8. For further granularity, Appx. §A offers an exhaustive list of the prompts implemented in our experiments. Importantly, it should be noted that all prompts are deployed in a zero-shot manner, without any demonstrations or examples.

4 LLM Robustness to Structural Perturbations

This section explores how LLMs interpret varied table structures in response to our first research question (§1). We probe the impact of three table perturbations on LLM performance (§4.1), uncover LLMs' challenges and limitations for direct table transposition and recoganize tranposed tables (§4.2), and introduce a structure normalization strategy (Norm) to mitigate these issues (§4.3).

4.1 Impacts of Table Perturbations on LLMs

In §3.1, we present three types of structural table perturbations: transposition ($\mathcal{T}^\top$), row-shuffling

[2] Column shuffling was not employed as the typical number of columns is limited and this shuffling had minimal impact on accuracy (Zhao et al., 2023).

Perturbation	DP	PyAgent
Original ($\mathcal{T}$)	59.50	55.91
+Shuffle ($\mathcal{T}_{\Pi}$)	52.21 -12.25%	47.91 -14.31%
+Transpose ($\mathcal{T}^{\top}$)	51.14 -14.05%	12.45 -77.73%
+Transpose&Shuffle ($\mathcal{T}_{\Pi}^{\top}$)	37.51 -36.96%	8.96 -83.97%

Table 1: Accuracy of GPT-3.5 under different table perturbations using Direct Prompting (DP) and Python Shell Agent (PyAgent).

LLMs As	Task Description	Accuracy
Transposer	$f(\mathcal{T}) \to \mathcal{T}^{\top}$	53.68
	$f(\mathcal{T}^{\top}) \to \mathcal{T}$	51.07
Detector	$f(\mathcal{T}) \to 0$	93.35
	$f(\mathcal{T}^{\top}) \to 1$	32.54
Determinator	$f(\mathcal{T}, \mathcal{T}_{0,*}, \mathcal{T}_{*,0}) \to \mathcal{T}_{0,*}$	97.39
	$f(\mathcal{T}^{\top}, \mathcal{T}_{0,*}, \mathcal{T}_{*,0}) \to \mathcal{T}_{*,0}$	94.77

Table 2: Evaluation results of GPT-3.5 on the 421 distinct tables of the WIKITABLEQUESTIONS (WTQ) dataset, covering three tasks: (1) *Transposer*, which involves switching between original ($\mathcal{T}$) and transposed ($\mathcal{T}^{\top}$) tables directly; (2) *Detector*, which identifies the need for table transposition (0 for no transposition, 1 for transposition required); and (3) *Determinator*, which chooses probable table headings either from the first row ($\mathcal{T}0, *$) or the first column ($\mathcal{T}*, 0$).

($\mathcal{T}_{\Pi}$), and their combination ($\mathcal{T}_{\Pi}^{\top}$). As demonstrated in Tab. 1, both reasoning methods, DP and PyAgent, exhibit significant performance declines, with more pronounced when transposition is applied. DP consistently outperforms PyAgent largely across perturbations, indicating that textual reasoning tends to be more resilient to these structural changes. This resilience can be attributed to LLMs' ability to grasp semantic connections and meanings irrespective of structural shifts. In contrast, symbolic reasoning, exemplified by PyAgent, is heavily reliant on table structure, making it more vulnerable, especially to transposition.

4.2 Limitations of Table Transposition with LLMs

To better understand LLMs' capabilities with regard to table structures, we investigate their ability on detecting tables in need of transposition and performing table transposition.

LLMs as Transposition Detectors. Given a table $\mathcal{T}$, the goal is to detect whether a table should be transposed for better comprehension by LLMs. This is formulated as a binary classification task:

$$f(\mathcal{T}) \to 0, \quad f(\mathcal{T}^{\top}) \to 1,$$

Where 0 denotes 'no need of transposition' and 1 indicates 'transposition needed'. Tab. 2 shows the results using the prompt in Appx. §A.4. GPT-3.5 correctly classified 93.35% of original tables $\mathcal{T}$ as not requiring transposition. However, its accuracy dramatically decreased to 32.54% on transposed tables $\mathcal{T}^{\top}$. Our observations highlight that LLMs suffer from structural bias in the interpretation of table orientations, predominantly leading to recommendations against transposition.

LLMs as Table Transposers. The objective is to switch between original and transposed table formats. Specifically, the goal is to directly yield $\mathcal{T}^{\top}$ given $\mathcal{T}$, and vice versa. Formally, the task is:

$$f(\mathcal{T}) \to \mathcal{T}^{\top}, \quad f(\mathcal{T}^{\top}) \to \mathcal{T}$$

We observed that GPT-3.5's proficiency in this task is limited, with an accuracy of 53.68% transposing row tables and 51.07% for the inverse operation, suggesting that LLMs can not transpose tables precisely. For a detailed error case study and further analysis, refer to the Appx. §B.

4.3 Table Structure Normalization

In addressing structural variations in tables, our goal is to ensure consistent interpretation and utility across diverse table structures. To normalize various table structures into well-ordered row-tables prior to downstream tasks, we introduce NORM, which is a two-stage normalization strategy: the first stage detects column-tables and transposing them into row-tables, while the second stage sorts the row-tables for enhanced comprehensibility. Through this approach, NORM accommodates for structural perturbations without compromising the understanding of the standardized row-tables.

Content-Aware Transposition Determination In the straightforward methods mentioned in §4.2, LLMs are affected by the loss of structure information of the table. Our approach aims to reduce this structural dependence by introducing a content-aware determination process, which leverages the semantic reasoning capabilities of LLMs, instead of perceiving the table's structure. Specifically, we analyze the inherent content within the first row ($\mathcal{T}_{0,*}$) and the first column ($\mathcal{T}_{*,0}$) of a given table

Method	$\mathcal{T}$	$\mathcal{T}_{\Pi}$	$\mathcal{T}^{\top}$	$\mathcal{T}_{\Pi}^{\top}$
DP	59.50	52.21	51.14	37.51
+NORM	58.66 -1.41%	58.66 +12.35%	58.30 +14.00%	57.71 +53.85%
PyAgent	55.91	47.91	12.43	8.96
+NORM	56.87 +1.72%	57.11 +19.20%	55.44 +346.02%	55.08 +514.73%

Table 3: Accuracy of GPT-3.5 under different table perturbations for Direct Prompting (DP) and Python Shell Agent (PyAgent) with NORM applied.

($\mathcal{T}$) to decide which is more semantically fitting to serve as the table's heading. This content-aware approach can be mathematically modeled as:

$$\begin{cases} f(\mathcal{T}, \mathcal{T}_{0,*}, \mathcal{T}_{*,0}) \rightarrow \mathcal{T}_{0,*} \\ f(\mathcal{T}^{\top}, \mathcal{T}_{0,*}, \mathcal{T}_{*,0}) \rightarrow \mathcal{T}_{*,0} \end{cases}$$

Here, a selection of the first row suggests that the current table structure is preferred, whereas opting for the first column signifies a need for transposition. The prompt detailing this method is provided in Appx. §A.5. Results in Tab. 2 highlight capability of GPT-3.5 in discerning table headings semantically, with accuracies of 97.39% and 94.77% respectively for original table and tranposed table.

Row Reordering. Upon transposition, our next objective is to ensure the logical coherence of the table data through reordering the rows. We instruct LLMs to suggest improved reordering strategies using the prompts as detailed in Appx. §A.6. Due to the subjective nature involved in identifying the most suitable order of a tabular data, and given that there are no widely recognized standards for this process, the effectiveness of the proposed sorting strategy will be evaluated based its downstream impact on the results of table QA task. We notice that when the entire well-ordered table is exposed, GPT-3.5 occasionally suggests alternative sorting strategies, leading to unnecessary complexity. To counteract this tendency and ensure a better sorting proposal, we strategically present the model with only the first three and the last three rows of the table. This selective exposure typically allows the model to discern logical ordering patterns without being influenced by existing table configurations.

Tab. 3 underscores the efficacy of NORM when applied prior to the two reasoning methods – DP and PyAgent. Demonstrably, NORM robustly mitigates structural perturbations, optimizing table comprehensibility for LLMs. The results illustrate that applying NORM does not detrimentally affect the original results ($\mathcal{T}$), and it effectively refines perturbated data, aligning the outcomes closely with the original results, and in some instances, even showing slight improvement. This suggests that NORM as a preprocessing step for preparing tabular data can enhance robust analysis by LLMs.

In addressing our initial research question, the analysis indicates that **LLMs' performance is sensitive to table structural variations**, with significant struggles observed in accurately interpreting the same tabular content under transposition and shuffling. While **textual reasoning demonstrates some resilience** to structural variations, **symbolic reasoning is significantly impacted**, particularly with transposed tables. **The NORM strategy effectively navigates these challenges** by eliminating dependency on table structures, providing consistent interpretation across diverse table structures without compromising the integrity or meaning of the original content.

5 Comparing Textual and Symbolic Reasoning

In this section, we delve into the comparison of textual and symbolic reasoning methods in LLMs for tabular data understanding (§5.1), further conducting a detailed error analysis (§5.2) to address the second research question (§1). We evaluate the performance of each reasoning strategy using GPT-3.5, shedding light on their strengths and challenges. In §4.3, we explored NORM to mitigate structural perturbations, enhancing generalized LLM performance and successfully restoring perturbed tables to accuracy levels similar to their original states. Therefore, subsequent analyses will exclusively consider the original tables ($\mathcal{T}$).

5.1 Results

Tab. 4 showcases the performance of GPT-3.5 when employed for both direct textual reasoning using DP and interactive symbolic reasoning using PyAgent. By instructing the model with the CoT (Wei et al., 2023) reasoning strategy to *think step by step, and then give the final answer*, as detailed in Appx. §A.1, we can achieve an accuracy of 58.66%. This surpasses the StructGPT's Iterative Reading-then-Reasoning method, which concentrates reasoning tasks by continually collecting relevant evidence. For tables with limited tokens, symbolic reasoning via PyAgent offers an accuracy of 56.87%,

Method	Accuracy
Few-shot Prompting Methods	
BINDER★ (Cheng et al., 2023)	63.61
BINDER♠ (Cheng et al., 2023)	55.07
DATER w/o SC★ (Ye et al., 2023)	61.75
DATER w/ SC★ (Ye et al., 2023)	68.99
Zero-shot Prompting Methods	
STRUCTGPT♠ (Jiang et al., 2023a)	51.77
NORM+DP♠	58.66
NORM+PYAGENT♠	56.87
NORM+PYAGENT-OMITTED♠	52.45
NORM+DP&PYAGENT w/ EVAL♠	64.22
DP w/ SC♠	66.39
+NORM♠	64.10
+NORM w/o RESORT♠	66.99
PYAGENT w/ SC♠	61.39
+NORM♠	63.77
+NORM w/o RESORT♠	62.84
DP&PYAGENT w/ MIX-SC♠	73.06
+NORM♠	72.40
+NORM w/o RESORT♠	73.65

Table 4: Performance comparison of various methods on a sampled subset ($\mathcal{T}$) of the WikiTableQuestions dataset. Methods marked with ★ are based on Codex, while those marked with ♠ are based on GPT-3.5. SC stands for self-consistency, and the results for SC-based methods are obtained by averaging over 100 shuffles to handle cases of ties during majority voting. In the Mix-SC method, answers from DP are prioritized over those from PyAgent due to DP's superior performance. This prioritization is followed in all experiments involving SC. NORM w/o RESORT refers to the NORM method without the reordering stage.

which is slightly behind the accuracy by DP in a single attempt. A distinct advantage of symbolic reasoning is its ability to only present parts of the table in the prompt. As our experiments revealed, after excluding the central rows and showcasing only the initial and final three rows, we manage to maintain an accuracy of 52.45% with a 4.42% drop compared to the full-table PyAgent results. This makes it possible to deal with larger tables with numerous rows using LLMs with limited context window. In the following sections, we will present a comprehensive analysis of the discrepancies and errors observed across these methods.

5.2 Error Analysis

To elucidate the challenges and limitations of DP and PyAgent, this section presents an in-depth error analysis by sampling 50 erroneous outputs for each. Tab. 5 summarizes the predominant error types for DP and PyAgent methods. *Table interpretation errors* significantly afflict the DP method, comprising 42% of its total errors, highlighting substantial challenges for LLMs in accurately interpreting table data. PyAgent primarily struggles with *coding errors*, constituting 38% of its total errors. These errors either originate from misunderstandings of table content, often overlooking subtle details, or manifest as inherent deficiencies in coding capabilities. These prevalent errors underscore the intrinsic challenges and limitations each method faces in the reasoning process. Detailed case studies on each error type are delineated in Appx. §C.

In response to the second research question, the analysis indicates **DP marginally surpasses PyAgent within single attempts**. Despite this, PyAgent can handle larger tables by processing partial table views. Notably, **DP encounters difficulties in accurate table interpretation**, while **PyAgent reveals instability in coding capabilities**.

6 Reasoning Aggregation

This section examines how combining multiple reasoning pathways can boost LLMs' accuracy in interpreting tabular data, which is in response to the third research question (§1).

6.1 Methods

Self-Consistency. Previous work has highlighted the advantages of generating multiple outputs from LLMs and adopting the most frequent answer, a mechanism known as self-consistency (SC; Wang et al. 2023). Tab. 4 showcases the notable improvements realized through self-consistency (aggregating 10 outputs), with DP achieving an accuracy of 64.84% and PyAgent attaining 63.49%.

Self-Evaluation. Based on our error analysis in §5.2, different reasoning methods excel at specific tasks. For instance, symbolic reasoning tends to outperform textual reasoning in counting and column localization tasks. To optimize the choice between these methods, we strategically use a prompt (referenced in Appx. §A.7), which avoids directly validating answers against tables but guides the LLM to choose between the two reasoning approaches based on the question's nature and each answer's clarity. By weighing the problem against the known strengths and weaknesses of each reasoning strategy, this tactic mitigates potential bias

Error Types	DP	PyAgent	Description	Case Study
Table Misinterpretation	42%	-[†]	LLMs incorrectly interpret the content in tables.	Appx. §C.1.1, Appx. §C.1.2
Coding Errors	-	38%	LLMs produce inaccurate code, typically due to issues with minor details.	Appx. §C.2.1, Appx. §C.2.2, Appx. §C.2.3
Misalignment Issue	24%	28%	Outputs are conceptually correct but the answers do not align with the instructions.	Appx. §C.3.1, Appx. §C.3.2
Logical Inconsistency	20%	10%	LLMs exhibit failures in reasoning, leading to contradictions or inconsistencies.	Appx. §C.4.1, Appx. §C.4.2
Execution Issue	-	12%	Issues emerge related to the execution of Python code.	Appx. §C.5.1, Appx. §C.5.2
Resorting Issue	10%	8%	The resorting stage in NORM changes the answers of some sequence-dependent questions.	Appx. §C.6

Table 5: Categorization of error types for the DP and PyAgent methods. [†]The *table interpretation errors* is not explicitly used for PyAgent, as these errors are included under *coding errors* to avoid overlap in categorization. The percentages for each reasoning method may not sum up to 100%, as the remaining percentage points are attributed to *other errors*, such as issues with dataset labeling, which are not categorized in our analysis.

towards textual reasoning by LLMs and enhances answer accuracy. As evidenced by Tab. 4, using self-evaluation boosts accuracy to 64.99%. Impressively, this method, using only two reasoning paths, matches the performance of using 10 paths of DP or PyAgent independently.

Mix Self-Consistency. According to §5.1, symbolic and textual reasoning exhibit distinct focuses but deliver similar performance. Consequently, we introduce *Mix Self-Consistency*, a method that selects a predetermined number of outputs for each type of inference, aiming for self-consistency. This approach hinges on the idea that multiple outputs can reflect the confidence levels of LLMs in answer generation. In scenarios where LLMs are less proficient, they tend to produce a diverse set of answers. Conversely, for tasks that LLMs handle adeptly, consistent answers are often generated across multiple reasoning attempts, converging towards one answer. Such convergence allows for the aggregation of model outputs that align with areas where LLMs exhibit stronger reasoning capabilities, thereby substantially improving accuracy. The detailed mechanics of how this approach is implemented within the framework of *Mix Self-Consistency*, including the aggregation and interpretation of these outputs, are further elucidated in Appx. §D.2.

Tab. 4 demonstrates that using mix self-consistency, which involves generating 5 outputs per inference type (totaling 10), enhances performance substantially, achieving an impressive accuracy of 72.40%, which achieves SOTA performance on the sampled WTQ data. The choice of generating 5 outputs per inference type (5+5) is a hyperparameter selection influenced by the dataset's distribution. We conducted an ablation study regarding this in Appx. §D.1. The decision to use an equal split (5+5) is based on the observed comparable performance between the two reasoning strategies.

6.2 Overall Evaluation

To evaluate our method thoroughly, we conduct a comprehensive pass of testing using the complete WTQ test set, integrating both NORM and Mix self-consistency mechanisms. Since re-sorting may change the answers of row index-related questions, we perform NORM without resorting in this evaluation. Originally, the NORM process included a re-sorting step to counteract the row-shuffling perturbation. However, as explored in the error analysis (§5.2) with a detailed case study in Appx. §C.6, re-sorting may inadvertently alter answers reliant on the initial sequence. Despite this limitation, it is noteworthy that re-sorting can be advantageous for questions not dependent on row indexes, particularly when dealing with tables that are initially unorganized or messy.

As illustrated in Tab. 6, our proposed method exhibits outstanding efficacy with an accuracy of 73.6%, significantly outperforming existing models to achieve SOTA performance on the complete WTQ test set. Importantly, our approach is conducted in a fully zero-shot manner.

To further analyze the impact of table size on method performance, we segmented the row num-

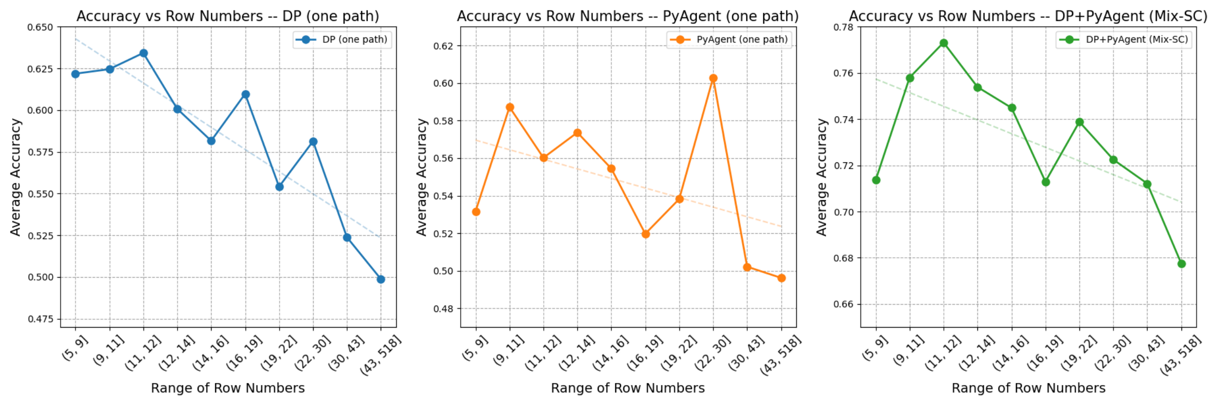

Figure 3: The impact of table size on the performance of DP, PyAgent, and Mix-SC (Combining 5 DP and 5 PyAgent) methods on the WikiTableQuestions test set. The horizontal axis represents the number of rows in the tables, divided into ranges, while the vertical axis denotes the average accuracy achieved by each method within the corresponding table size range.

Method	**Accuracy**
Fine-tuning Based Models	
TAPAS (Herzig et al., 2020)	48.8
T5-3B (Xie et al., 2022)	49.3
TAPAX (Liu et al., 2022)	57.5
REASTAP (Zhao et al., 2022)	58.7
OMNITAB (Jiang et al., 2022)	63.3
LLMs Based Methods	
STRUCTGPT★ (Jiang et al., 2023a)	48.4
BINDER★ (Cheng et al., 2023)	55.5
BINDER♠ (Cheng et al., 2023)	64.6
LEVER♠ (Ni et al., 2023)	65.8
DATER♠ (Ye et al., 2023)	65.9
Ours★	**73.6**

Table 6: Comparison of various methods on all test data of WTQ. ★ denotes methods based on the GPT-3.5 (OpenAI, 2023a); ♠ denotes methods based on the Codex (OpenAI, 2022). Results are directly sourced from the referenced paper.

bers into 10 ranges, each containing approximately 430 data points, and calculated the average accuracy within these intervals. Fig. 3 visualizes the average accuracy across these ranges for each method. It is evident that there is a shared trend of diminishing accuracy as the number of rows increases, suggesting that all methods are subject to decreased efficacy in the context of long tables.

The decline in performance with larger tables can be attributed to the complexity of handling long-context data and the abundance of potentially interfering information, often resulting in an increased error rate. The insights gained from this analysis point towards a need for the development of better symbolic methods for handling long tables, which might be capable of effectively narrowing down the scope of larger tables, either by selective attention to relevant segments or by intelligently summarizing the data, to mitigate the challenges posed by long-context information.

In response to the third research question, our findings reveal that **reasoning path aggregation significantly enhances LLMs' accuracy in table reasoning tasks**. Notably, **the *Mix Self-Consistency* method achieves an accuracy of 73.6% on the WTQ dataset, surpassing the previous SOTA by a considerable margin**. The *Self-Evaluation* strategy also contributes to this remarkable performance by adeptly selecting between reasoning approaches.

7 Conclusion

This study investigated the proficiency of LLMs in tabular reasoning. The findings suggest that LLMs are sensitive to the structural variance of tables, but the application of the proposed normalization strategy can stabilize table structures and improve resistance to structural perturbations. When comparing reasoning approaches, textual reasoning demonstrated a slight advantage over symbolic reasoning, with each strategy exhibiting unique strengths. Furthermore, integrating multiple reasoning strategies via mix self-consistency proved beneficial for overall interpretation accuracy, surpassing previous SOTA results on the WIKITABLEQUESTIONS dataset. These observations contribute to the understanding of LLMs' capabilities in tabular reasoning and provide insights for further improvements.

Limitation

While this study provides insights into tabular data reasoning with LLMs, it is pertinent to acknowledge its limitations. First, the exclusive utilization of GPT-3.5, due to the budgetary constraints, may limit the generalizability of our findings, as exploration with GPT-4 might offer enhanced outcomes. Second, all table data are sourced from Wikipedia, which may introduce potential data leakage or memorization issues, as certain answers might be implicitly available within the LLMs' training data, thus potentially biasing results. Lastly, several perturbation-sensitive table-based questions, especially regarding table perturbations like shuffling, may impact the precision of the reported accuracy, as demonstrated answers may change based on the structural modifications of the table.

Acknowledgment

We appreciate the reviewers for their insightful comments and suggestions. Fei Wang is supported by the Annenberg Fellowship and the Amazon ML Fellowship. Muhao Chen is supported by the NSF Grant IIS 2105329, the NSF Grant ITE 2333736, the Faculty Startup Fund of UC Davis, a Cisco Research Award and two Amazon Research Awards.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, and Bing Xiang. 2023. Dr.spider: A diagnostic evaluation benchmark towards text-to-sql robustness.

Harrison Chase. 2022. LangChain.

Wenhu Chen. 2023. Large language models are few(1)-shot table reasoners.

Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W Cohen. 2020. Open question answering over tables and text. In *International Conference on Learning Representations*.

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. *ICLR*, abs/2210.02875.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

Leo Gao, John Schulman, and Jacob Hilton. 2022. Scaling laws for reward model overoptimization.

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. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4971–4983, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings.

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*. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023a. Structgpt: A general framework for large language model to reason on structured data.

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. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 932–942, Seattle, United States. Association for Computational Linguistics.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. Active retrieval augmented generation.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. 2023a. Sheetcopilot: Bringing software productivity to the next level through large language models.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023b. Starcoder: may the source be with you!

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023c. Making large language models better reasoners with step-aware verifier.

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.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. Starcoder 2 and the stack v2: The next generation.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented language models: a survey.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. Webgpt: Browser-assisted question-answering with human feedback.

Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*.

OpenAI. 2022. Codex.

OpenAI. 2023a. Chatgpt.

OpenAI. 2023b. Dall·e 3.

OpenAI. 2023c. Gpt-4 technical report.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context.

Significant Gravitas. 2023. Auto-GPT.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2022. Learning to summarize from human feedback.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *ArXiv preprint*, abs/2302.13971.

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. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Fei Wang, Kexuan Sun, Jay Pujara, Pedro Szekely, and Muhao Chen. 2021. Table-based fact verification with salience-aware learning. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4025–4036, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Fei Wang, Zhewei Xu, Pedro Szekely, and Muhao Chen. 2022. Robust (controlled) table-to-text generation with structure-aware equivariance learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

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

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

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. *EMNLP*.

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, 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*, pages 8413–8426, Online. Association for Computational Linguistics.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2023a. Tablellama: Towards open large generalist models for tables.

Wenqi Zhang, Yongliang Shen, Weiming Lu, and Yueting Zhuang. 2023b. Data-copilot: Bridging billions of data and humans with autonomous workflow.

Yilun Zhao, Linyong Nan, Zhenting Qi, Rui Zhang, and Dragomir Radev. 2022. ReasTAP: Injecting table reasoning skills during pre-training via synthetic reasoning examples. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9006–9018, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yilun Zhao, Chen Zhao, Linyong Nan, Zhenting Qi, Wenlin Zhang, Xiangru Tang, Boyu Mi, and Dragomir Radev. 2023. Robut: A systematic study of table qa robustness against human-annotated adversarial perturbations.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-most prompting enables complex reasoning in large language models.

Appendices

A Prompts

A.1 Prompt of Direct Prompting (DP).

```
You are an advanced AI capable of analyzing and understanding information within tables. Read the
    table below regarding "[TITLE]".

[TABLE]

Based on the given table, answer the following question:

[QUESTION]

Let's think step by step, and then give the final answer. Ensure the final answer format is only
    "Final Answer: AnswerName1, AnswerName2..." form, no other form. And ensure the final answer is
    a number or entity names, as short as possible, without any explanation.
```

A.2 Prompt of Python Agent.

```
You are working with a pandas dataframe in Python. The name of the dataframe is `df`. Your task is
    to use `python_repl_ast` to answer the question posed to you.

Tool description:
- `python_repl_ast`: A Python shell. Use this to execute python commands. Input should be a valid
    python command. When using this tool, sometimes the output is abbreviated - ensure it does not
    appear abbreviated before using it in your answer.

Guidelines:
- **Aggregated Rows**: Be cautious of rows that aggregate data such as 'total', 'sum', or 'average'.
    Ensure these rows do not influence your results inappropriately.
- **Data Verification**: Before concluding the final answer, always verify that your observations
    align with the original table and question.

Strictly follow the given format to respond:

Question: the input question you must answer
Thought: you should always think about what to do to interact with `python_repl_ast`
Action: can **ONLY** be `python_repl_ast`
Action Input: the input code to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: after verifying the table, observations, and the question, I am confident in the final
    answer
Final Answer: the final answer to the original input question (AnswerName1, AnswerName2...)

Notes for final answer:
- Ensure the final answer format is only "Final Answer: AnswerName1, AnswerName2..." form, no other
    form.
- Ensure the final answer is a number or entity names, as short as possible, without any explanation.
- Ensure to have a concluding thought that verifies the table, observations and the question before
    giving the final answer.

You are provided with a table regarding "[TITLE]". This is the result of `print(df.to_markdown())`:

[TABLE]

**Note**: All cells in the table should be considered as `object` data type, regardless of their
    appearance.

Begin!
Question: [QUESTION]
```

A.3 Prompt of LLMs as Table Transposer

```
You are given the following table:
```

```
[TABLE]

Please transpose this table. Maintain the format I give, with each row beginning with '|' and each
    cell separated by ' | '. Do not change the content of any cell. Your response should solely
    consist of the transposed table, without any additional text.
```

A.4 Prompt of LLMs as Table Transposition Detector

```
Please examine the provided table:

[TABLE]

To enhance readability and facilitate efficient data analysis, it is often suggested that the table
    headings be horizontally located in the first/topmost row.

Please evaluate the table with this consideration in mind, and provide your response in the
    following format:

**Table Headings**: List the headings of the table, separated by commas.
**Table Evaluation**: Identify whether the headings listed are horizontally located in the
    first/topmost row. If not, describe the position.
**Transpose Recommended**: Indicate if transposing is recommended. Answer with only "YES" or "NO",
    without any additional explanation.
```

A.5 Prompt of Content-Aware Transposition Determination

```
You are an advanced AI capable of analyzing and understanding information within tables. Read the
    table below regarding "[TITLE]".

[TABLE]

Headings of a table are labels or titles given to rows or columns to provide a brief description of
    the data they contain.

Based on the given table, the headings of the table are more likely to be:

(A) [FIRST_ROW]
(B) [FIRST_COLUMN]
(C) None of the above

Directly give your choice. Ensure the format is only "Choice: (A)/(B)/(C)" form, no other form,
    without any explanation.
```

A.6 Prompt of Resorting

```
You are an advanced AI capable of analyzing and understanding information within tables. Read the
    table below regarding "[TITLE]":

[TABLE]

Note: Only selected rows from the beginning and end of the table are displayed for brevity.
    Intermediate rows are omitted and represented by "..." for clarity.

The table column headings are provided below, separated by semicolons:

[HEADINGS]

In order to optimize the interpretability and readability of the data, follow these guidelines to
    determine the most suitable sorting method:

Sorting Guidelines:

1. Evaluate columns based on data types such as numerical, alphabetical, chronological, categorical,
    or other relevant sorting methods.
2. Identify any patterns or relationships in the data that would be highlighted by certain sorting
    methods.
3. Consider column position, as those on the left may sometimes have sorting priority.
4. If applicable, consider sorting by multiple columns in a prioritized sequence.
```

```
Provide your decision using one of the following statements:

- For sorting using a single column: "Sort by: [Name of Column]".
- For sorting using multiple columns: "Sort by: [Primary Column Name], [Secondary Column Name], ...".
- If no specific sorting seems advantageous: "Sort by: N/A".

Your response should strictly follow the formats provided.
```

A.7 Prompt of Self-Evaluation

```
Below is a markdown table regarding "[TITLE]":

[TABLE]

You're tasked with answering the following question:

[QUESTION]

You have 2 answers derived by two different methods. Answer A was derived by prompting the AI to
    think step-by-step. Answer B was derived by interacting with a Python Shell.

Answer A is [COT_ANSWER].
Answer B is [AGENT_ANSWER].

Your task is to determine which is the correct answer. It is crucial that you strictly adhere to the
    following evaluation process:

1. **Preliminary Evaluation**: Begin by evaluating which of the two answers directly addresses the
    question in a straightforward and unambiguous manner. A direct answer provides a clear response
    that aligns closely with the query without introducing additional or extraneous details. If one
    of the answers is not a direct response to the question, simply disregard it.
2. **Nature of the Question**: If both answers appear to be direct answers, then evaluate the nature
    of the question. For tasks involving computation, counting, and column-locating, especially
    when for extensive table, the Python Shell (Answer B) might be more precise. However, always
    remain cautious if the Python Shell's output appears off (e.g., error messages, success
    notifications, etc.). Such outputs may not be trustworthy for a correct answer.
3. **Final Verdict**: Finally, after thorough evaluation and explanation, provide your verdict
    strictly following the given format:
  - Use "[[A]]" if Answer A is correct.
  - Use "[[B]]" if Answer B is correct.

Note:
1. Each method has its own strengths and weaknesses. Evaluate them with an unbiased perspective.
    When in doubt, consider the nature of the question and lean towards the method that is most
    suited for such queries.
2. Ensure that your verdict is provided after evaluation, at the end.
```

B Analysis for LLMs as Table Transposer

B.1 Case Study

Original Table

Year	1931	Spring 1932	Fall 1932	Spring 1933	1933/34	1934/35	...	1951/52	1952/53	1953/54	1954/55	1955/56
Division	1	1	1	1	N/A	N/A	...	N/A	N/A	N/A	N/A	N/A
League	ASL	ASL	ASL	ASL	ASL	ASL	...	ASL	ASL	ASL	ASL	ASL
Reg. Season	6th (Fall)	5th?	3rd	?	2nd	2nd	...	6th	6th	1st	8th	6th
Playoffs	No playoff	No playoff	No playoff	?	No playoff	No playoff	...	No playoff	No playoff	Champion (no playoff)	No playoff	No playoff
National Cup	N/A	1st Round	N/A	Final	?	?	...	?	Semifinals	Champion	?	?

Ground Truth

Year	Division	League	Reg. Season	Playoffs	National Cup
1931	1	ASL	6th (Fall)	No playoff	N/A
Spring 1932	1	ASL	5th?	No playoff	1st Round
Fall 1932	1	ASL	3rd	No playoff	N/A
Spring 1933	1	ASL	?	?	Final
1933/34	N/A	ASL	2nd	No playoff	?
1934/35	N/A	ASL	2nd	No playoff	?
1935/36	N/A	ASL	1st	Champion (no playoff)	?
1936/37	N/A	ASL	5th, National	Did not qualify	Champion
1937/38	N/A	ASL	3rd(t), National	1st Round	?
1938/39	N/A	ASL	4th, National	Did not qualify	?
1939/40	N/A	ASL	4th	No playoff	?
1940/41	N/A	ASL	6th	No playoff	?
1941/42	N/A	ASL	3rd	No playoff	?
1942/43	N/A	ASL	6th	No playoff	?
1943/44	N/A	ASL	9th	No playoff	?
1944/45	N/A	ASL	9th	No playoff	?
1945/46	N/A	ASL	5th	No playoff	?
1946/47	N/A	ASL	6th	No playoff	?
1947/48	N/A	ASL	6th	No playoff	?
1948/49	N/A	ASL	1st(t)	Finals	?
1949/50	N/A	ASL	3rd	No playoff	?
1950/51	N/A	ASL	5th	No playoff	?
1951/52	N/A	ASL	6th	No playoff	?
1952/53	N/A	ASL	6th	No playoff	Semifinals
1953/54	N/A	ASL	1st	Champion (no playoff)	Champion
1954/55	N/A	ASL	8th	No playoff	?
1955/56	N/A	ASL	6th	No playoff	?

Model Prediction

Year	Division	League	Reg. Season	Playoffs	National Cup
1931	1	ASL	6th (Fall)	No playoff	N/A
Spring 1932	1	ASL	5th?	No playoff	1st Round
Fall 1932	1	ASL	3rd	No playoff	N/A
Spring 1933	1	ASL	?	?	Final
1933/34	N/A	ASL	2nd	No playoff	?
1934/35	N/A	ASL	2nd	No playoff	?
1935/36	N/A	ASL	1st	Champion (no playoff)	?
1936/37	N/A	ASL	5th, National	Did not qualify	Champion
1937/38	N/A	ASL	3rd(t), National	1st Round	?
1938/39	N/A	ASL	4th, National	Did not qualify	?
1939/40	N/A	ASL	4th	No playoff	?
1940/41	N/A	ASL	6th	No playoff	?
1941/42	N/A	ASL	3rd	No playoff	?
1942/43	N/A	ASL	6th	No playoff	?
1943/44	N/A	ASL	9th	No playoff	?
1944/45	N/A	ASL	9th	No playoff	?
1945/46	N/A	ASL	5th	No playoff	?
1946/47	N/A	ASL	6th	No playoff	?
1947/48	N/A	ASL	6th	No playoff	?
1948/49	N/A	ASL	1st(t)	Finals	?
1949/50	N/A	ASL	3rd	No playoff	?
1950/51	N/A	ASL	5th	No playoff	?
1951/52	N/A	ASL	6th	No playoff	Semifinals
1952/53	N/A	ASL	6th	No playoff	Champion
1953/54	N/A	ASL	1st	Champion (no playoff)	?
1954/55	N/A	ASL	8th	No playoff	?
1955/56	N/A	ASL	6th	No playoff	?

Figure 4: An example error case of content misalignment occurring within cells when leveraging LLMs directly to transpose a table. **Blue Table (Top):** The original table subjected to a transposition operation. **Green Table (Buttom Left):** The ground truth table subsequent to transposition. **Purple Table (Buttom Right):** GPT-3.5's output of transposed table. Cells erroneously aligned or displaced are highlighted in red.

Fig. 4 illustrates a typical mistake made by LLMs when transposing tables, a problem that becomes more evident when a table has many identical or similar entries. Take, for example, the 'Nation Cup' column shown in the figure, which is filled with numerous ? symbols. LLMs, limited in processing structured data, often mishandle such tables, leading to misplacements or misalignments. This highlights the fundamental difficulties and limitations LLMs face in accurately transposing tables containing repetitive or similar data cells.

B.2 Analysis

A further examination of the results, as shown in Fig. 5, illustrates that transposition accuracy for LLMs as direct table transposer is associated with the table's dimensions. The accuracy in row-to-column transposition ($\mathcal{T} \rightarrow \mathcal{T}^{\top}$) is distinctly sensitive to the original table's row count, whereas column-to-row

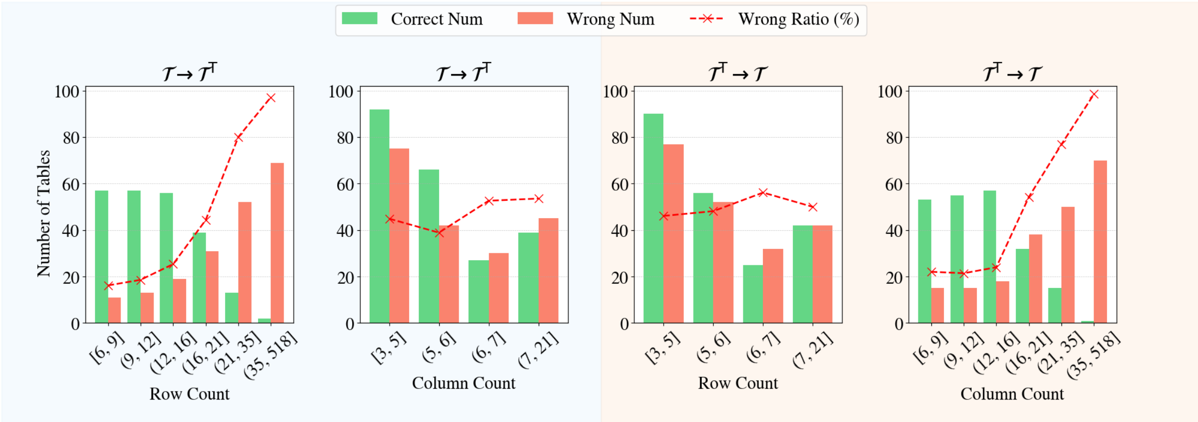

Figure 5: Performance of GPT-3.5 as direct table transposer: from original to transposed tables ($\mathcal{T} \to \mathcal{T}^\top$) and from transposed to original tables ($\mathcal{T}^\top \to \mathcal{T}$), with different row and column counts.

transposition ($\mathcal{T}^\top \to \mathcal{T}$) accuracy is similarly related on the number of columns. This observation can be potentially attributed to the inherent characteristics and organizational structure of table data. In most of the row tables, cells within a given column often display homogeneous data types, such as numerical or temporal values. This homogeneity can pose significant challenges for LLMs, as the models might struggle to differentiate between semantically similar cells during the transposition process, thereby leading to potential misalignments and misplacements, particularly as the number of rows increases. Conversely, in those column tables, cells within a row may exhibit similar data types, introducing analogous challenges and potential errors during transposition.

C Error Case Study for WTQ

C.1 Table Misinterpretation

C.1.1 Counting Error

Speed skating at the 2006 Winter Olympics - Men's 1000 metres

	Rank	Pair	Name	Country	Time	Time behind
...	...	...	...	...	...	...
9	10	6	Dmitry Dorofeyev	Russia	1:09.74	+0.85
10	11	19	Jeremy Wotherspoon	Canada	1:09.76	+0.87
11	12	9	Beorn Nijenhuis	Netherlands	1:09.85	+0.96
...	...	...	...	...	...	...
14	15	13	Alexey Proshin	Russia	1:10.14	+1.25
15	16	7	François-Olivier Roberge	Canada	1:10.20	+1.31
16	17	11	Choi Jae-bong	South Korea	1:10.23	+1.34
17	18	12	Petter Andersen	Norway	1:10.38	+1.38
18	19	17	Denny Morrison	Canada	1:10.44	+1.44
19	20	15	Yusuke Imai	Japan	1:10.48	+1.59
...	...	...	...	...	...	...
26	27	14	Takaharu Nakajima	Japan	1:11.10	+2.21
27	28	3	Takahiro Ushiyama	Japan	1:11.21	+2.32
28	29	12	Steven Elm	Canada	1:11.36	+2.47
29	30	8	Maurizio Carnino	Italy	1:11.44	+2.55
...	...	...	...	...	...	...
39	–	2	Ermanno Ioriatti	Italy	DQ	–
40	–	6	Erik Zachrisson	Sweden	DQ	–

What is the name of the last manufacturer on this chart?

To determine the number of finishers Canada had in the Men's 1000 metres speed skating event at the 2006 Winter Olympics, we need to identify the rows in the table that correspond to Canadian athletes who completed the race.

From the table, we can see that the Canadian athletes are listed in rows 10, 15, 18, 23, 24, 27, 28, and 36. These rows correspond to the following athletes: Jeremy Wotherspoon, François-Olivier Roberge, Denny Morrison, Mun Jun, Janne Hänninen, Takahiro Ushiyama, Steven Elm, and Risto Rosendahl.

Therefore, the number of finishers Canada had is 8.

Final Answer: 8 ❌

Correct Answer: 4

Figure 6: Example of a *table misinterpretation* error by DP, where the LLMs make mistakes attributable to its deficiency in performing counting tasks effectively.

Fig. 6 highlights a typical error related to table comprehension, emphasizing a common problem in LLMs when dealing with tasks that involve statistical analysis. It points to a weakness in LLMs' ability to accurately process and respond to questions based on statistical information without the help of external tools.

C.1.2 Locating Error

2005-06 FIS Ski Jumping World Cup

	Rank	Name	Nationality	1st (m)	2nd (m)	Points
0	11	Jakub Janda	CZE	123.5	128.5	252.6
1	5	Andreas Kofler	AUT	126.5	132	261.3
2	4	Adam Małysz	POL	127	130.5	261.5
3	3	Janne Ahonen	FIN	127	130.5	264
4	2	Tami Kiuru	FIN	128.5	130.5	264.7
5	1	Matti Hautamäki	FIN	132	131.5	274.3

Question: Who has the top 2nd score?

To determine who has the top 2nd score, we need to find the highest value in the "2nd (m)" column of the table.

By analyzing the table, we can see that the highest value in the "2nd (m)" column is 132, achieved by both Andreas Kofler and Matti Hautamäki.

Therefore, the top 2nd score is 132 achieved by Andreas Kofler and Matti Hautamäki.

Final Answer: Andreas Kofler, Matti Hautamäki ❌

Correct Answer: Andreas Kofler

Figure 7: Example of a *table misinterpretation* error by DP, where the model fails accurately locating the specific cell.

Fig. 7 showcases a common error in table interpretation associated with LLMs. This error originates from the LLMs' linearization process, which impairs their ability to recognize table structures. Although the model efficiently identifies the highest value, `132`, in the `2nd(m)` column, it inaccurately associates this value with the `1nd(m)` column, assuming it represents the same feature as in the `2nd(m)` column. This leads to a misplacement of the value in the table's interpretation.

C.2 Coding Error

C.2.1 Attribute Noise Error

List of The Citadel Bulldogs football seasons

	Season	Team	Record	Head Coach	Quarterback	...	Runner Up
0	1970	Dallas Cowboys	10–4	Tom Landry*	Craig Morton	...	San Francisco 49ers
1	1971	Dallas Cowboys†	11–3	Tom Landry*	Roger Staubach*	...	San Francisco 49ers
2	1972	Washington Redskins	11–3	George Allen*	Billy Kilmer	...	Dallas Cowboys
3	1973	Minnesota Vikings	12–2	Bud Grant*	Fran Tarkenton*	...	Dallas Cowboys
4	1974	Minnesota Vikings	10–4	Bud Grant*	Fran Tarkenton*	...	Los Angeles Rams
5	1975	Dallas Cowboys	10–4	Tom Landry*	Roger Staubach*	...	Los Angeles Rams
6	1976	Minnesota Vikings	11–2–1	Bud Grant*	Fran Tarkenton*	...	Los Angeles Rams
7	1977	Dallas Cowboys†	12–2	Tom Landry*	Roger Staubach*	...	Minnesota Vikings
8	1978	Dallas Cowboys	12–4	Tom Landry*	Roger Staubach*	...	Los Angeles Rams
9	1979	Los Angeles Rams	9–7	Ray Malavasi	Pat Haden	...	Tampa Bay Buccaneers
...	...	...	...	...	...	...	...
21	1991	Washington Redskins†	14–2	Joe Gibbs*	Mark Rypien	...	Detroit Lions
22	1992	Dallas Cowboys†	13–3	Jimmy Johnson	Troy Aikman*	...	San Francisco 49ers
23	1993	Dallas Cowboys†	12–4	Jimmy Johnson	Troy Aikman*	...	San Francisco 49ers
24	1994	San Francisco 49ers†	13–3	George Seifert	Steve Young*	...	Dallas Cowboys
25	1995	Dallas Cowboys†	12–4	Barry Switzer	Troy Aikman*	...	Green Bay Packers
26	1996	Green Bay Packers†	13–3	Mike Holmgren	Brett Favre	...	Carolina Panthers
...	...	...	...	...	...	...	...
43	2013	Seattle Seahawks†	13–3	Pete Carroll	Russell Wilson	...	San Francisco 49ers

How many times did the cowboys win the nfc championship in the 1970s?

Thought: To answer this question, I need to count the number of times the Cowboys won the NFC championship in the 1970s. I can do this by filtering the dataframe based on the team and season columns.

Action: Use `python_repl_ast` to filter the dataframe and count the number of rows.

Action Input: `df[(df['Team'] == 'Dallas Cowboys') & (df['Season'] >= 1970) & (df['Season'] < 1980)].shape[0]`

Observation:

```
3
```

Final Answer: 3 ✗

Correct Answer: 5

Figure 8: Example of a *Coding Issue* error by PyAgent, where the abnormal variant data entries leads to inaccurate output.

Fig. 8 illustrates a minor issue stemming from a coding mistake. The table shown in the figure features a variety of data under the `Team` column. Alongside the expected `Dallas Cowboy` entries, there are cells with a slight variation: `Dallas Cowboy`†. The Python Shell Agent used failed to recognize these unusual variations. This is evident from the use of the `df[‘Team’] = “Dallas Cowboy”` command for calculating occurrences, leading to a discrepancy in the final count and resulting in inaccurate outcomes.

C.2.2 Special Row Misinterpretation Error

Nicolás Terol

	Season	Class	Moto	Races	Win	Podiums	Pole	Pts	Position
0	2004	125cc	Aprilia	1	0	0	0	0	NC
1	2005	125cc	Derbi	13	0	0	0	1	36th
2	2006	125cc	Derbi	16	0	0	0	53	14th
3	2007	125cc	Derbi	17	0	0	0	19	22nd
4	2008	125cc	Aprilia	17	1	5	0	176	5th
5	2009	125cc	Aprilia	16	1	4	0	179.5	3rd
6	2010	125cc	Aprilia	16	3	14	1	296	2nd
7	2011	125cc	Aprilia	16	8	11	7	302	1st
8	2012	Moto2	Suter	17	0	1	0	37	17th
9	2013	Moto2	Suter	17	3	4	1	150	7th
10	2014	Moto2	Suter	1	0	0	0	0*	NC*
11	Total	nan	nan	147	16	39	9	1213.5	nan

How many seasons has nicolás won at least 10 races?

Thought: To answer this question, I need to count the number of seasons where Nicolás has won at least 10 races. I can do this by filtering the dataframe based on the condition that the 'Win' column is greater than or equal to 10, and then counting the number of unique seasons.

Action: python_repl_ast

Action Input: `df[df['Win'] >= 10]['Season'].nunique()`

Observation:

```
1
```

Thought: The observation indicates that Nicolás has won at least 10 races in 1 season.

Final Answer: 1 ❌

Correct Answer: 0

Figure 9: Example of a *coding error* by PyAgent, where PyAgent misinterpretes the special row – `Total`.

Fig. 9 displays an error where the Python Shell Agent incorrectly interprets a special row, treating the `Total` row as a valid season entry. In this case, while calculating the seasons in which Nicolás won at least 10 races, the Agent did not properly exclude the cumulative `Total` row in its code for computation. Consequently, it erroneously reported one season as meeting the criteria.

Comparison of Linux distributions

	Distribution	...	x86	x86-64	...	arm	s390x	alpha	m68k
0	OES2-Linux	...	Yes	Yes	...	No	No	No	No
1	Arch Linux	...	Yes (>=i686)	Yes	...	No	No	No	No
...	...	...	...	...	...	...	...	...	...
9	Distribution	...	x86	x86-64	...	arm	s390x	alpha	m68k
10	Fedora	...	Yes	Yes	...	No	Yes	No	No
...	...	...	...	...	...	...	...	...	...
32	XBMC	...	Yes	No	...	No	No	No	No
33	Yellow Dog Linux	...	No	No	...	No	No	No	No
34	openSUSE	...	Yes	Yes	...	No	No	No	No

How many distributions support the x86 architecture?

🤔**Thought**: To answer this question, I need to count the number of distributions that support the x86 architecture.

🛠**Action**: I will use the `value_counts()` method on the 'x86' column to count the number of occurrences of 'Yes'.

🧑‍💻**Action Input**: `df['x86'].value_counts()`

💻**Observation**:

```
x86
Yes                          28
No                            4
Yes (>=i686)                  1
x86                           1
Discontinued\n2.1-6           1
Name: count, dtype: int64
```

🤔**Thought**: The `value_counts()` method has counted the number of occurrences of each unique value in the 'x86' column. Now I need to sum the counts of 'Yes' to get the total number of distributions that support the x86 architecture.

🛠**Action**: I will sum the counts of 'Yes' from the previous observation.

🧑‍💻**Action Input**: `28 + 1 + 1`

💻**Observation**:

```
30
```

💡**Final Answer**: 30 ❌

Correct Answer: 29

Figure 10: Example of another *coding error* by PyAgent, where the PyAgent misinterpretes the special row which is a nested heading.

Fig. 10 depicts an error where the Python Shell Agent incorrectly includes a special row in its calculations. Specifically, when counting the number of Linux distributions supporting the x86 architecture, the agent erroneously counts a nested heading row. As indicated in the figure, the row indexed at 9 is not a valid data entry but rather serves as a nested heading for the table. This row should have been excluded from the count, resulting in an inaccurate calculation (29 $\rightarrow$ 30) of distributions supporting the x86 architecture.

C.2.3 Incorrect Coding

Stoewer

	Type	Construction period	Cylinder	Capacity	Power	Vmax
0	10 PS (7 kW; 10 hp)	1901–1902	straight-2	1.527 cc	18 PS (13,2 kW)	50 km/h (31 mph)
1	8/14 PS	1902–1905	straight-2	1.527 cc	14 PS (10,3 kW)	50 km/h (31 mph)
2	20 PS (15 kW; 20 hp)	1904–1905	straight-4	7.946 cc	45 PS (33 kW)	85 km/h (53 mph)
3	P4 (11/22 PS)	1905–1910	straight-4.	3.054 cc	22 PS (16,2 kW)	70 km/h (43 mph)
...	...	...	...	...	...	...
19	D9 (8/32 PS)	1923–1924	straight-4	2.290 cc	32 PS (23,5 kW)	90 km/h (56 mph)
20	D12 (12/45 PS)	1923–1924	straight-6	3.107 cc	45 PS (33 kW)	100 km/h (62 mph)
...	...	...	...	...	...	...
40	Greif Junior	1936–1939	flat-4	1.484 cc	34 PS (25 kW)	100 km/h (62 mph)
41	Arkona	1937–1940	straight-6	3.610 cc	80 PS (59 kW)	120 km/h (75 mph)–140 km/h (87 mph)
42	Sedina	1937–1940	straight-4	2.406 cc	55 PS (40 kW)	110 km/h (68 mph)

How many cars had a maximum velocity of at least 100 km/h?

🤔**Thought**: To answer this question, I need to count the number of cars that had a maximum velocity of at least 100 km/h. I can do this by filtering the 'Vmax' column and counting the number of rows that meet the condition.

🛠**Action**: I will filter the 'Vmax' column for cars that have a maximum velocity of at least 100 km/h and then count the number of rows.

Action Input: `df[df['Vmax'].str.contains('100 km/h')].shape[0]`

💻**Observation**:

```
7
```

🤔**Thought**: After filtering the 'Vmax' column for cars with a maximum velocity of at least 100 km/h, I found that there are 7 cars that meet this condition.

💡**Final Answer**: 7 ❌

Correct Answer: 17

Figure 11: Example of a *coding error* by PyAgent, where the coding is incorrect

Fig. 11 shows an issue that originates from a basic coding mistake. Although the agent correctly grasps the concept, aiming to filter for entries with a `maximum velocity of at least 100 km/h`, it falters in the implementation stage by using a `contains("100 km/h")` statement in the code. This error is akin to the *Logical Inconsistency* error described in (DP), where a mismatch between the understanding of a concept and its practical execution becomes evident.

C.3 Misalignment Issue

C.3.1 Answer Format Issue

1984 IMSA GT Championship season

	Rnd	Circuit	GTP Winning TeamGTP Winning Drivers	GTO Winning TeamGTO Winning Drivers	GTU Winning TeamGTU Winning Drivers	Results
0	1	Daytona	#00 Kreepy Krauly Racing	#4 Stratagraph Inc.	#76 Malibu Grand Prix	Results
...	...	...	...	...	...	...
9	5	Riverside	#56 Blue Thunder Racing	#38 Mandeville Auto Tech	#87 Performance Motorsports	Results
10	6	Laguna Seca	#56 Blue Thunder Racing	#77 Brooks Racing	#99 All American Racers	Results
11	6	Laguna Seca	Randy Lanier	John Bauer	Jim Adams	Results
...	...	...	...	...	...	...
21	11	Portland	Bill Whittington\n Randy Lanier	David Schroeder\n Tom Hendrickson	Jack Baldwin	Results
22	12	Sears Point	#56 Blue Thunder Racing	#77 Brooks Racing	#98 All American Racers	Results
23	12	Sears Point	Bill Whittington	John Bauer	Dennis Aase	Results
...	...	...	...	...	...	...
33	17	Daytona	#14 Holbert Racing	#67 Roush Racing	#87 Performance Motorsports	Results

What number did Brooks Racing have?

Thought: To find the number of Brooks Racing, I need to filter the dataframe based on the team name and then extract the number.
Action: Use pandas filtering to find the row with the team name "Brooks Racing" and extract the number.
Action Input: `df[df['GTP Winning TeamGTP Winning Drivers'].str.contains('Brooks Racing')]['GTP Winning TeamGTP Winning Drivers']`
Observation:

```
Series([], Name: GTP Winning TeamGTP Winning Drivers, dtype: object)
```

Thought: It seems that there is no row in the dataframe where the GTP Winning TeamGTP Winning Drivers column contains the name "Brooks Racing". I should double-check the spelling and formatting of the team name.
Action: Print the unique values in the GTP Winning TeamGTP Winning Drivers column to verify the spelling and formatting.
Action Input: `df['GTP Winning TeamGTP Winning Drivers'].unique()`
Observation:

```
['#00 Kreepy Krauly Racing'
'Sarel van der Merwe\\n Graham Duxbury\\n Tony Martin'
'Doc Bundy\\n Brian Redman' '#04 Group 44'
'Mauricio DeNarvaez\\n Hans Heyer\\n Stefan Johansson'
'#48 DeNarvaez Enterprises' '#16 Marty Hinze Racing' 'Don Whittington'
'Don Whittington\\n Randy Lanier' '#56 Blue Thunder Racing'
'Randy Lanier' 'Bill Whittington\\n Randy Lanier' 'Sarel van der Merwe'
'Al Holbert\\n Derek Bell' '#14 Holbert Racing'
'Al Holbert\\n Jim Adams\\n Derek Bell' 'Bill Whittington'
'#57 Blue Thunder Racing' 'Dale Whittington\\n Randy Lanier']
```

Thought: The unique values in the GTP Winning TeamGTP Winning Drivers column do not include "Brooks Racing". It seems that there is no row in the dataframe where the team name is "Brooks Racing". I should double-check the column name to ensure I am searching in the correct column.
Action: Print the column names to verify the column name.
Action Input: `df.columns`
Observation:

```
Index(['Rnd', 'Circuit', 'GTP Winning TeamGTP Winning Drivers',
    'GTO Winning TeamGTO Winning Drivers',
    'GTU Winning TeamGTU Winning Drivers', 'Results'],
    dtype='object')
```

Thought: The column name is 'GTP Winning TeamGTP Winning Drivers', not 'GTO Winning TeamGTO Winning Drivers'. I should modify the filtering condition to search for 'Brooks Racing' in the correct column.
Action: Use pandas filtering to find the row with the team name "Brooks Racing" and extract the number.
Action Input: `df[df['GTO Winning TeamGTO Winning Drivers'].str.contains('Brooks Racing')]['GTO Winning TeamGTO Winning Drivers']`
Observation:

```
10    #77 Brooks Racing
22    #77 Brooks Racing
Name: GTO Winning TeamGTO Winning Drivers, dtype: object
```

Final Answer: Brooks Racing had the number 77. ✅

Correct Answer: 77

Figure 12: Example of a *misalignment issue* by PyAgent, where the answer format does not follow the instruction in the prompt to give a parsable answer. Note that emojis presented in the figure are purely for visual aid and were not incorporated in actual experiments; the same applies to any figures below.

Fig. 12 displays a scenario in which the final answer produced did not align with the specified prompt instructions. In this instance, the Python Shell Agent correctly executed a series of interactions and accurately identified the final answer as `77`. However, the response given was `Brooks Racing had the number 77.`. Although the reasoning and the result are correct within their respective contexts, the format of the response impedes the parsing of the correct answer.

C.3.2 Answer Deviation Error

2007–08 Washington Huskies men's basketball team

	#	Name	Height	Weight (lbs.)	Position	Class	Hometown	Previous Team(s)
0	21	Artem Wallace	6'8"	250	C	Jr.	Toledo, WA, U.S.	Toledo HS
1	32	Joe Wolfinger	7'0"	255	C	RS So.	Portland, OR, U.S.	Northfield Mount Hermon School
2	22	Justin Holiday	6'6"	170	F	Fr.	Chatsworth, CA, U.S.	Campbell Hall School
3	24	Quincy Pondexter	6'6"	210	F	So.	Fresno, CA, U.S.	San Joaquin Memorial HS
4	40	Jon Brockman	6'7"	255	F	Jr.	Snohomish, WA, U.S.	Snohomish Sr. HS
5	44	Darnell Gant	6'8"	215	F	Fr.	Los Angeles, CA, U.S.	Crenshaw HS
6	11	Matthew Bryan-Amaning	6'9"	235	F	Fr.	London, England, U.K.	South Kent School
7	1	Venoy Overton	5'11"	180	G	Fr.	Seattle, WA, U.S.	Franklin HS
8	5	Justin Dentmon	5'11"	185	G	Jr.	Carbondale, IL, U.S.	Winchendon School
9	20	Ryan Appleby	6'3"	170	G	Sr.	Stanwood, WA, U.S.	Florida
10	0	Joel Smith	6'4"	210	G	RS Jr.	Lompoc, CA, U.S.	Brewster Academy
11	4	Tim Morris	6'4"	210	G	Sr.	Spokane Wa, U.S.	Central Valley HS

Are there at least 13 names on the chart?

To determine if there are at least 13 names on the chart, we need to count the number of rows in the table.

Step 1: Count the number of rows in the table.

There are 12 rows in the table.

Step 2: Compare the count to 13.

The count of 12 is less than 13.

Final Answer: 12

Correct Answer: no

Figure 13: Example of a *misalignment issue* by DP, where the final answer does not directly answer the question.

Fig. 13 shows a DP example where the final answer does not directly address the posed question. Faced with the binary query `Are there at least 13 names on the chart?`, the anticipated response should be a simple yes or no. However, the LLM responds with `12`. Although this answer aligns conceptually with the data in the table and the logic seems accurate, it does not conform to the direct and binary response format required by the question.

List of defunct National Basketball Association teams

	Team	City	Years active	Seasons played	Win–loss record	Win%	Playoffs appearances
0	Detroit Falcons	Detroit, Michigan	1946–1947	1	20–40	.333	0
1	Toronto Huskies	Toronto, Ontario	1946–1947	1	22–38	.367	0
2	Pittsburgh Ironmen	Pittsburgh, Pennsylvania	1946–1947	1	15–45	.250	0
3	Cleveland Rebels	Cleveland, Ohio	1946–1947	1	30–30	.500	1
4	Providence Steamrollers	Providence, Rhode Island	1946–1949	3	46–122	.274	0
5	St. Louis Bombers	St. Louis, Missouri	1946–1950	4	122–115	.515	3
6	Chicago Stags	Chicago, Illinois	1946–1950	4	145–92	.612	4
7	Washington Capitols	Washington, D.C.	1946–1951	5	157–114	.579	4
8	Baltimore Bullets*	Baltimore, Maryland	1947–1954	8	158–292	.351	3
9	Indianapolis Jets	Indianapolis, Indiana	1948–1949	1	18–42	.300	0
...	...	...	...	...	...	...	...
16	BAA Indianapolis	Indianapolis, Indiana	Never Played	0	0–0	N/A	0

Which team had the same number of playoffs appearances as the St. Louis Bombers?

🤔**Thought**: To find the team that had the same number of playoffs appearances as the St. Louis Bombers, I need to compare the "Playoffs appearances" column of each team with the value of the St. Louis Bombers.

🛠**Action**: Filter the dataframe to get the row corresponding to the St. Louis Bombers.

🧑‍💻**Action Input**: `df[df['Team'] == 'St. Louis Bombers']`

💻**Observation**:

```
                 Team                 City Years active  Seasons played Win–loss record   Win%  Playoffs appearances
5  St. Louis Bombers  St. Louis, Missouri    1946–1950               4         122–115  0.515                     3
```

🤔**Thought**: Now that I have the row for the St. Louis Bombers, I can extract the number of playoffs appearances.

🛠**Action**: Access the value of the "Playoffs appearances" column for the St. Louis Bombers.

🧑‍💻**Action Input**: `df[df['Team'] == 'St. Louis Bombers']['Playoffs appearances']`

💻**Observation**:

```
5    3
Name: Playoffs appearances, dtype: int64
```

💡**Final Answer**: 3 ❌

Correct Answer: Baltimore Bullets

Figure 14: Example of a *misalignment issue* by PyAgent, where the final answer does not directly answer the question.

Fig. 14 illustrates a case with the Python Shell Agent where the final response fails to directly answer the posed question. The question `Which team has the same number of playoffs appearances as the St. Louis Bombers?` clearly requests the identification of a specific team. Yet, it is noted that the agent prematurely delivers an answer upon finding data related to the playoff appearances of the St. Louis Bombers. While the direction of the python shell agent's reasoning appears correct, the resultant answer ultimately falls short of resolving the question correctly.

C.4 Logical Inconsistency

C.4.1 Reasoning Conflict in DP

Chief of the National Guard Bureau

	Number	Name	Service	From	To
0	1	COL Erasmus M. Weaver, Jr.	USA	February 14, 1908	March 14, 1911
1	2	BG Robert K. Evans	USA	March 15, 1911	August 31, 1912
...	...	...	...	...	...
8	8	MG William G. Everson	USA	October 1, 1929	November 30, 1931
9	9	MG George E. Leach	USA	December 1, 1931	November 30, 1935
10	10	MG Albert H. Blanding	USA	January 31, 1936	January 30, 1940
11	11	MG John F. Williams	USA	January 31, 1940	January 30, 1944
12	12	MG Butler B. Miltonberger	USA	February 1, 1946	September 29, 1947
13	13	MG Kenneth F. Cramer	USA	September 30, 1947	September 4, 1950
14	14	MG Raymond H. Fleming	USA	August 14, 1951	February 15, 1953
15	15	MG Edgar C. Erickson	USA	June 22, 1953	May 31, 1959
...	...	...	...	...	...
39	(Acting)	MG John R. D'Araujo, Jr.	USA	August 1, 1994	September 30, 1994
40	(Acting)	MG Raymond F. Rees	USA	August 4, 2002	April 10, 2003

Question: Was george e. leach or kenneth f. cramer the chief of the national guard bureau in 1934?

To determine whether George E. Leach or Kenneth F. Cramer was the Chief of the National Guard Bureau in 1934, we need to find the individuals who held the position during that time period.

From the table, we can see that George E. Leach served as the Chief of the National Guard Bureau from December 1, 1931, to November 30, 1935. Kenneth F. Cramer, on the other hand, served as the Chief of the National Guard Bureau from September 30, 1947, to September 4, 1950.

Since neither George E. Leach nor Kenneth F. Cramer held the position in 1934, the answer is neither.

Final Answer: Neither ❌

Correct Answer: George E. Leach

Figure 15: Example of a *logical inconsistency* by DP where a problem with logical reasoning leads to a reasoning conflict in the context.

Fig. 15 presents an example of a *Logical Inconsistency* error occurring during the interpretation of tabulated data. The error in reasoning is occurred in determining whether George E. Leach or Kenneth F. Cramer was the Chief of the National Guard Bureau in 1934. The reasoning text accurately states that `George E. Leach served from December 1, 1931, to November 30, 1935`, and `Kenneth F. Cramer served from September 30, 1947, to September 4, 1950`. Despite this, the interpretation erroneously concludes that neither was in the role in 1934, leading to a contradiction between the information and the final answer.

C.4.2 Reasoning Mistakes in PyAgent

Wiseman hypothesis

	Tablet	Genealogy	Narrative	Colophon
0	1	Creation of Universe 1:1	1:2 to 2:3	"This is the account of the heavens and of the earth when they were created." 2:4
1	2	Heavens and Earth 2:4	2:5 to 4:26	"This is the written account of Adam." 5:1
2	3	Adam to Noah 5:1 - 32	6:1 - 8	"This is the account of Noah." 6:9
...	...	...	...	...
8	9	Abraham to Isaac 25:19	25:20 to 35:29	"This is the account of Esau." 36:1 (eldest son)
9	10	Descendants of Esau 36:2 - 5	36:6 - 8	"This is the account of Esau." 36:9
10	11	Descendants of Esau 36:10 to 37:1	no narrative	"This is the account of Jacob." 37:2
11	nan	no genealogy	37:2 to 50:26	no colophon

Tables 9 and 10 are each accounted by who?

🤔**Thought**: To answer this question, I need to find the rows corresponding to Tablets 9 and 10 and check the values in the "Genealogy" column.

🛠**Action**: Filter the dataframe to include only the rows with Tablets 9 and 10.

🧑‍💻**Action Input**: `df[(df['Tablet'] == 9) | (df['Tablet'] == 10)]`

💻**Observation**:

```
   Tablet                   Genealogy       Narrative                                    Colophon
8     9.0          Abraham to Isaac 25:19  25:20 to 35:29  "This is the account of Esau." 36:1 (eldest son)
9    10.0  Descendants of Esau 36:2 - 5        36:6 - 8                "This is the account of Esau." 36:9
```

🤔**Thought**: The rows corresponding to Tablets 9 and 10 are accounted by Abraham and the descendants of Esau, respectively.

💡**Final Answer**: Abraham, Descendants of Esau ❌

Correct Answer: Esau

Figure 16: Example of a *logical inconsistency* error by PyAgent where the LLMs make mistakes in reasoning on the observations.

Fig. 16 depicts an instance of a *Logical Inconsistency Error* during the Python Shell interaction. In this case, LLMs successfully the necessary steps to gather information about `Tablet 9` and `Tablet 10`, a misinterpretation of the retrieved data results in flawed reasoning. This leads to an error in the conclusion drawn from the interaction.

C.5 Execution Issue

C.5.1 Interaction Bound or Looping Error

Comparison of Linux distributions

	#	Date	Venue	Opponent	Score	Result	Competition
0	1	27 March 1999	Mestalla, Valencia, Spain	Austria	3–0	9–0	Euro 2000 qualifying
1	2	27 March 1999	Mestalla, Valencia, Spain	Austria	5–0	9–0	Euro 2000 qualifying
2	3	31 March 1999	Olimpico, Serravalle, San Marino	San Marino	0–3	0–6	Euro 2000 qualifying
3	4	8 September 1999	Vivero, Badajoz, Spain	Cyprus	1–0	8–0	Euro 2000 qualifying
4	5	8 September 1999	Vivero, Badajoz, Spain	Cyprus	2–0	8–0	Euro 2000 qualifying
5	6	8 September 1999	Vivero, Badajoz, Spain	Cyprus	4–0	8–0	Euro 2000 qualifying
6	7	26 January 2000	Cartagonova, Cartagena, Spain	Poland	2–0	3–0	Friendly
7	8	26 January 2000	Cartagonova, Cartagena, Spain	Poland	3–0	3–0	Friendly

Which team competed for the Euro 2000 qualifying the most consecutive years?

🤔**Thought**: To find the team that competed for the Euro 2000 qualifying the most consecutive years, I need to count the number of consecutive years each team participated in the Euro 2000 qualifying.
🛠**Action**: I will group the dataframe by the "Opponent" column and count the number of unique years in the "Date" column for each opponent.
🧑‍💻**Action Input**: `df.groupby('Opponent')['Date'].apply(lambda x: x.dt.year.nunique())`
💻**Observation**:

```
AttributeError: Can only use .dt accessor with datetimelike values
```

🤔**Thought**: The error message suggests that the "Date" column is not in a datetime format. I need to convert the "Date" column to a datetime format before using the .dt accessor.
🛠**Action**: I will convert the "Date" column to a datetime format using the `pd.to_datetime()` function.
🧑‍💻**Action Input**: `df['Date'] = pd.to_datetime(df['Date'])`
💻**Observation**:

```
success!
```

🤔**Thought**: Now that the "Date" column is in a datetime format, I can proceed with grouping the dataframe and counting the number of unique years for each opponent.
🛠**Action**: I will group the dataframe by the "Opponent" column and count the number of unique years in the "Date" column for each opponent.
🧑‍💻**Action Input**: `df.groupby('Opponent')['Date'].apply(lambda x: x.dt.year.nunique())`
💻**Observation**:

```
Opponent
Austria       1
Cyprus        1
Poland        1
San Marino    1
Name: Date, dtype: int64
```

🤔**Thought**: The output shows the number of unique years each opponent participated in the Euro 2000 qualifying. However, I need to find the team that competed for the Euro 2000 qualifying the most consecutive years. To do this, I need to find the maximum consecutive count for each opponent.
🛠**Action**: I will modify the lambda function to calculate the maximum consecutive count for each opponent.
🧑‍💻**Action Input**: `df.groupby('Opponent')['Date'].apply(lambda x: x.diff().dt.days.ne(1).cumsum().value_counts().max())`
💻**Observation**:

```
Opponent
Austria       1
Cyprus        1
Poland        1
San Marino    1
Name: Date, dtype: int64
```

🤔**Thought**: The output is not what I expected. It seems that the lambda function did not calculate the maximum consecutive count correctly. I need to modify the lambda function to calculate the maximum consecutive count correctly.
🛠**Action**: I will modify the lambda function to calculate the maximum consecutive count correctly.
🧑‍💻**Action Input**: `df.groupby('Opponent')['Date'].apply(lambda x: x.diff().dt.days.ne(1).cumsum().value_counts().max() + 1)`
💻**Observation**:

```
Opponent
Austria       2
Cyprus        2
Poland        2
San Marino    2
Name: Date, dtype: int64
```

Correct Answer: Cyprus

Figure 17: An example of an *execution issue* by PyAgent, where the agent attempts to fix an coding error but falls into a loop.

Fig. 17 demonstrates an instance of *Execution Issue* error made by the Python Shell Agent. In the process of identifying the team that participated in Euro 2000 qualifying for the most consecutive years, the agent faces difficulties in the data processing phase. Initially, an error occurs due to the `Date` column not containing `datetime` objects. Then the agent successfully converts the entries into the appropriate format. However, the agent, in trying to compute the number of consecutive participation years for each team, gets stuck in a loop of continually refining its calculation method without arriving at a conclusive answer within the given interaction steps.

C.5.2 Non-Observable Action Error

Stoewer

	Game	Date	Team	Score	...	LocationAttendance	Record
0	33	January 2	@ Detroit	L 92–98	...	The Palace of Auburn Hills\n22,076	8–25
1	34	January 3	@ Indiana	L 117–122	...	Conseco Fieldhouse\n12,765	8–26
2	35	January 5	@ New Jersey	L 90–98	...	Izod Center\n12,314	8–27
3	36	January 6	@ Chicago	L 94–99	...	United Center\n18,060	8–28
4	37	January 9	Miami	L 115–119 (OT)	...	ARCO Arena\n12,587	8–29
5	38	January 11	Dallas	W 102–95	...	ARCO Arena\n12,294	9–29
6	39	January 13	Orlando	L 107–139	...	ARCO Arena\n11,168	9–30
7	40	January 14	@ Golden State	W 135–133 (3OT)	...	Oracle Arena\n19,122	10–30
8	41	January 16	Milwaukee	L 122–129	...	ARCO Arena\n11,663	10–31
...	...	...	...	...	...	...	...
14	47	January 28	@ Boston	L 100–119	...	TD Banknorth Garden\n18,624	10–37
15	48	January 30	Chicago	L 88–109	...	ARCO Arena\n13,356	10–38

How many consecutive games were played in the ARCO Arena from January 2nd to January 24th?

🤔**Thought**: To answer this question, I need to find the consecutive games played in the ARCO Arena from January 2nd to January 24th. I can do this by filtering the dataframe based on the date and location, and then counting the number of rows.

🛠**Action**: Filter the dataframe based on the date and location.

🧑‍💻**Action Input**: `df_filtered = df[(df['Date'] >= 'January 2') & (df['Date'] <= 'January 24') & (df['LocationAttendance'].str.contains('ARCO Arena'))]`

💻**Observation**:

```
success!
```

🤔**Thought**: Now that I have filtered the dataframe, I can count the number of rows to determine the number of consecutive games played in the ARCO Arena from January 2nd to January 24th.

🛠**Action**: Count the number of rows in the filtered dataframe.

🧑‍💻**Action Input**: `consecutive_games_count = len(df_filtered)`

💻**Observation**:

```
success!
```

🤔**Thought**: Now that I have counted the number of rows in the filtered dataframe, I have the number of consecutive games played in the ARCO Arena from January 2nd to January 24th.

💡**Final Answer**: 4 ❌

Correct Answer: 3

Figure 18: Example of a *Execution Issue* error by PyAgent, where the agent inputs non-observable actions into Python Shell.

Fig. 18 illustrates a situation where PyAgent encounters a *Non-Observable Action Error*. Specifically, the actions given by the PyAgent, for example, `consecutive_games_count = len(df_filtered)`, do not generate any observable output in the Python Shell. Consequently, the agent does not receive any valid observations to aid it in deriving the correct answer, thus, leading to an incorrect count of consecutive games played in the ARCO Arena from January 2nd to January 24th.

C.6 Resorting Issue

Table Before Norm

	Year	Manufacturer	Model	Length (feet)	Quantity	Fleet Series	Fuel Propulsion	Powertrain
0	1996	New Flyer	D60	60 (articulated)	30	1901-1930*	Diesel	Detroit Diesel Series 50\nAllison B400R
1	1998	NABI	416	40	133	3001-3067, 3101-3166*	Diesel	Cummins M11E\nAllison B400R
...	...	...	...	...	...	...	...	...
19	2013	Gillig	Low-floor Advantage	40	65	1301-1365	Diesel	Cummins ISL 280 HP \nAllison B400 6-speed
20	2013	New Flyer	Xcelsior D60	60	23	2201-2223	Diesel	Cummins ISL 330 HP\nAllison B400 6-speed
21	2013	Gillig	Low-floor Advantage	40	55	6101-6155	Diesel	Cummins ISL 280 HP\nAllison B400 6-speed

Table After Norm (Sort by Year, Manufacturer, Model)

	Year	Manufacturer	Model	Length (feet)	Quantity	Fleet Series	Fuel Propulsion	Powertrain
0	1996	New Flyer	D60	60 (articulated)	30	1901-1930*	Diesel	Detroit Diesel Series 50\nAllison B400R
1	1998	NABI	416	40	133	3001-3067, 3101-3166*	Diesel	Cummins M11E\nAllison B400R
...	...	...	...	...	...	...	...	...
19	2013	Gillig	Low-floor Advantage	40	65	1301-1365	Diesel	Cummins ISL 280 HP \nAllison B400 6-speed
20	2013	Gillig	Low-floor Advantage	40	55	6101-6155	Diesel	Cummins ISL 280 HP\nAllison B400 6-speed
21	2013	New Flyer	Xcelsior D60	60	23	2201-2223	Diesel	Cummins ISL 330 HP\nAllison B400 6-speed

What is the name of the last manufacturer on this chart?

Answer before Norm (Ground Truth): Gillig

Answer after Norm: New Flyer

Figure 19: Example of a *Normalization Issue* error by DP, where the correct answer changes due to the resorting stage in Norm.

Fig. 19 shows a case of data inconsistency due to the application of the resorting stage in the Norm procedures. The figure's upper table displays the original format, with `Gillig` as the manufacturer in the final row. However, after resorting as suggested by LLMs, the lower table in the figure lists `New Flyer` as the last row's manufacturer. This change, while seemingly minor in the broader context of table comprehension, significantly impacts responses to specific queries like "`What is the name of the last manufacturer on the chart?`"

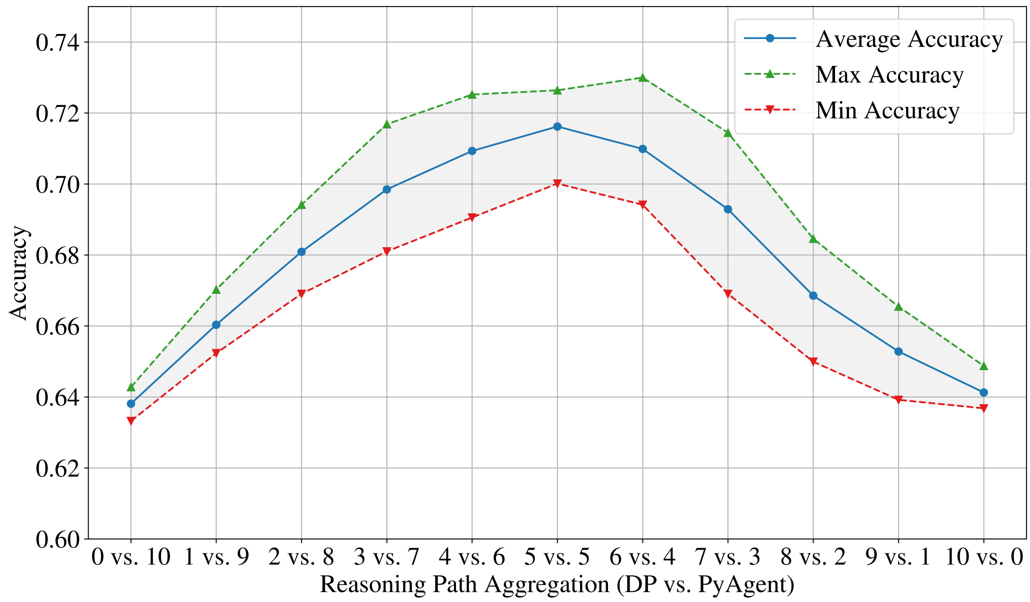

Figure 20: Accuracy results for the *Mix Self-Consistency* method applied to the sampled WTQ dataset, with varying combinations of DP and PyAgent outputs (depicted as DP vs. PyAgent on the x-axis). The combinations range from 10 DP vs. 0 PyAgent to 0 DP vs. 10 PyAgent. Each data point represents the maximum, minimum, and average accuracies obtained from 100 tests per combination, conducted using random sampling. Note that for the 10 DP vs. 0 PyAgent and 0 DP vs. 10 PyAgent combinations, there is no random sampling of paths. However, variance is observed due to the presence of multiple equally probable answer sets generated by the 10 paths, leading to different possible selections of answers even without sampling, thereby introducing randomness into the results.

D Analysis of Mix Self-Consistency

D.1 Ablation Study of Output Selection

This section presents an ablation study conducted to elucidate the effect of various combinations of DP and PyAgent outputs on the performance of the *Mix Self-Consistency* method. For this experiment, we systematically explored different combinations while keeping the total output count constant at ten. Each combination was tested 100 times through random shuffling. For each test, maximum, minimum, and average accuracies were recorded.

Fig. 20 shows the results of the ablation study. The 5+5 combination (5 DP + 5 PyAgent) consistently gives the highest minimum and average accuracies among all tested combinations, making it a robust and reliable choice for this task. The 4+6 combination (4 DP + 6 PyAgent) secured the highest maximum accuracy in our tests.

Through this ablation study, we aim to provide insights into how different output selections influence the effectiveness of the *Mix Self-Consistency* method. Importantly, the choice of output combination should be considered as a hyperparameter that is intimately related to the distribution of the dataset being used. Given that different reasoning strategies exhibit unique strengths and weaknesses, it is crucial to tailor the output combination to align with the characteristics of the specific tasks and datasets in question, thereby maximizing the performance of the *Mix Self-Consistency* method.

D.2 Mechanics of Mix Self-Consistency in Output Selection

The effectiveness of the *Mix Self-Consistency* method in achieving high accuracy largely stems from its ability to harness the strengths of different reasoning methods. Intuitively, the multiple outputs from certain reasoning method can be interpreted as the confidence score for the generated answers. In scenarios where a method excels, its outputs often tend to converge towards a common answer, signifying higher

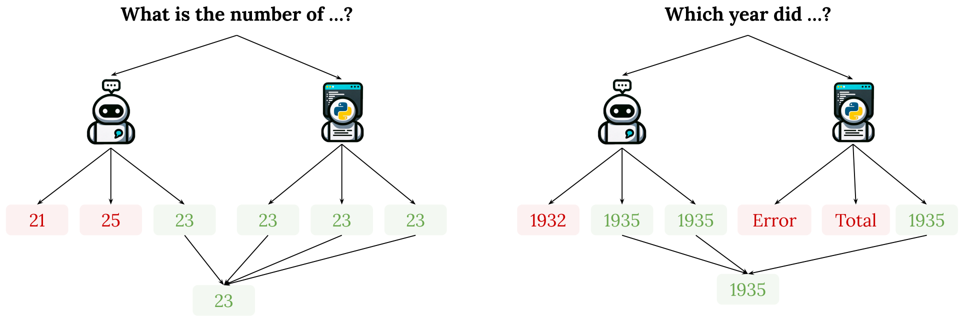

Figure 21: An illustration of *Mix Self-Consistency* by aggreagting outputs from multiple reasoning methods to form a unified, high-confidence prediction..

confidence and reliability. In contrast, a method less suited to the problem at hand tends to produce more diverse results, indicative of a lower level of confidence. By aggregating these outputs from different methods and applying majority voting, the *Mix Self-Consistency* method refines these variations into a more accurate prediction. As shown in Fig. 21, This process leverages the strengths of the employed reasoning methods, thereby enhancing overall performance.

E Results of Mix Self-Consistency on TabFact

This section presents the additional results of applying the *Mix Self-Consistency* method to the TabFact dataset, as part of an extended investigation to verify and evaluate the method's adaptability and effectiveness in other related tasks beyond WTQ dataset.

For TabFact, a subsample of 500 data points was randomly selected from the test set. The experimental setup mirrored that of the WTQ experiments, employing the same parameters such as temperature settings for model inference. The strategy for output selection in the TabFact experiment also follows the 5+5 combination, which proves to be the best for the WTQ dataset, to aggregate the output answers from 5 instances of DP and 5 instances of PyAgent. Additionally, all the prompts (e.g., DP, PyAgent) used in the TabFact experiment were slightly modified to align with the requirements of the fact-checking scenarios.

Method	Accuracy
StructGPT (Jiang et al., 2023a)	0.708
Dater (Ye et al., 2023)	0.874
Ours	**0.885**

Table 7: Accuracy results of different methods without fine-tuning on the TabFact dataset.

Tab. 7 summarizes the accuracy results of the *Mix Self-Consistency* method, StructGPT, and Dater on the TabFact dataset. *Mix Self-Consistency* can also achieve the highest accuracy, outperforming both StructGPT and Dater in fact-checking.