Improving Table Reasoning through Table Decomposition and Normalization

by

Md Mahadi Hasan Nahid

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Department of Computing Science
University of Alberta

Abstract

Table reasoning is a challenging task that requires understanding both natural language questions and structured tabular data. While Large Language Models (LLMs) have shown impressive capabilities in natural language understanding and generation, they often struggle with large tables due to their limited input length. Additionally, LLMs face challenges in tasks involving tabular data—especially those requiring symbolic reasoning—due to the structural variance and inconsistency in table cell values commonly found in web tables. In this thesis, we address these challenges with two novel approaches. First, we propose TabSQLify, a method that leverages Text-to-SQL generation to decompose tables into smaller, relevant sub-tables containing only essential information. This approach significantly reduces the input context length, making the task more scalable and efficient for large-scale table reasoning applications. Our evaluation on challenging datasets, including WikiTableQuestion and TabFact, demonstrates that TabSQLify achieves superior performance compared to prevailing methods and shows notable accuracy improvements, surpassing LLM-based baseline models.

In the second part of our study, we focus on enhancing the symbolic reasoning performance of LLMs when dealing with tabular data, particularly web tables with structural variance and inconsistency in cell values. We introduce NormTab, a framework for normalizing web tables as a one-time preprocessing step. This normalization improves consistency and structure, thereby supporting symbolic reasoning on tabular data. Our experimental evaluation on challenging datasets shows that NormTab significantly enhances symbolic reasoning performance, highlighting the importance

and effectiveness of table normalization in LLM-based reasoning tasks.

Preface

The main structure of this thesis is based on papers that are either published or under submission. Specifically, Chapter 3 is based on our paper published at NAACL 2024, while Chapter 4 is derived from our paper submitted to the EMNLP 2024 conference.

- Md Mahadi Hasan Nahid and Davood Rafiei. 2024. TabSQLify: Enhancing Reasoning Capabilities of LLMs Through Table Decomposition. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 5725–5737, Mexico City, Mexico. Association for Computational Linguistics. (published) [1]

- Md Mahadi Hasan Nahid and Davood Rafiei. 2024. NormTab: Improving Symbolic Reasoning in LLMs Through Tabular Data Normalization. arXiv preprint arXiv:2406.17961. (submittet) [2]

To my parents, with heartfelt gratitude for their endless support.

Acknowledgements

I extend my heartfelt gratitude to my supervisor, Davood Rafiei, for his unwavering guidance and exceptional support throughout my master's thesis journey. His insights, patience, and encouragement significantly shaped the direction of my research. I am also profoundly thankful to Dr. Greg Kondrak and Dr. Mario Nascimento, esteemed members of my thesis committee, for their valuable feedback and comments that enhanced the quality of my work.

I appreciate the esteemed members of the Database Research Group, whose knowledge sharing sessions broadened my understanding and focused my research. These interactions were crucial in guiding and strengthening my research pursuits.

I would like to extend my heartfelt gratitude to my parents, whose unwavering support has been the cornerstone of my academic journey. Their love and guidance have been instrumental in achieving this milestone. I am also deeply thankful to my beloved wife, Nishat Rezoana, for her constant support, patience, and understanding throughout the challenging process of completing this thesis. Her love, encouragement, and steadfast belief in me have been crucial to my success in this research journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Tables are universally recognized as foundational structures for organizing and presenting structured information across a wide array of domains. They play a pivotal role in diverse fields such as databases, spreadsheets, open data repositories, web pages, and document collections. In databases, tables serve as the fundamental units for storing and managing structured data, enabling efficient querying, updating, and retrieval of information. Spreadsheets utilize tables to organize data into rows and columns, facilitating tasks ranging from financial analysis to project management.

Moreover, tables are integral components of web pages and document collections, where they encapsulate information ranging from product specifications and pricing details to scientific research findings and historical records. This structured presentation enables users to quickly locate and extract pertinent information, enhancing usability and knowledge dissemination across digital platforms.

Overall, tables are indispensable tools for structuring and presenting information across various disciplines, ensuring data integrity, accessibility, and usability in both digital and physical formats. Their ubiquity underscores their critical role in modern information management and analysis.

1.1 Motivation

Table reasoning is the process of understanding and extracting meaningful information from tabular data to answer questions or perform specific tasks. It involves several key components, including data parsing, which focuses on extracting table elements like headers, rows, and cells; schema mapping, which helps understand the relationships between columns and rows; contextual understanding, where data is interpreted within the context of a specific query; and inference and computation, which involve making logical inferences and performing necessary calculations. The importance of table reasoning lies in its ability to handle structured information, commonly found in databases, spreadsheets, and web pages. This format is easy for humans to interpret, making it a valuable tool for extracting important insights from structured data. Moreover, table reasoning helps automate data processing and analysis, significantly reducing manual effort and saving time. Key downstream tasks that benefit from table reasoning include TableQA, which focuses on table-based question answering; Table-FV, which deals with table-based fact verification; and table summarization, which involves condensing information from tables into a more digestible format.

In table reasoning tasks, both textual reasoning and symbolic reasoning play crucial roles in extracting and interpreting information. Textual reasoning involves understanding and processing the natural language content within a table, such as interpreting the meanings of words, phrases, and sentences to answer questions or perform specific tasks. It leverages the linguistic context to make sense of the data presented in a tabular format. On the other hand, symbolic reasoning focuses on the logical and mathematical operations that can be performed on the data. This includes tasks like mapping relationships between columns and rows, executing SQL queries, performing calculations, and making logical inferences based on the structured nature of the data. Together, these reasoning approaches enable comprehensive table reasoning,

allowing models to handle both the semantic nuances of textual data and the formal, rule-based operations required for accurate data analysis. Textual reasoning focuses on understanding the semantic information in tables to directly generate answers, but it becomes challenging when dealing with large tables. On the other hand, symbolic reasoning, aided by programs like SQL or Python, excels in numerical operations and is scalable for large tables. However, it can be prone to errors when real-world tables are not clean. While textual reasoning directly processes the table and question to provide an answer, symbolic reasoning involves generating and executing code to derive the final response.

Developing natural language interfaces for tabular data poses a significant challenge, primarily in terms of effectively interpreting the semantics of table cells and understanding the relationships between cell values in response to a user query. This challenge is accentuated when tables are enveloped in text, such as titles, captions, and contextual text within a document. In these instances, the scope of reasoning expands beyond the confines of table cells to incorporate the surrounding natural language text. This reasoning is essential for many downstream tasks such as table-based fact verification and table-based question answering (TableQA). As depicted in Figure 1.1, table-based reasoning is intricate, demanding sophisticated textual, numerical, and logical reasoning across both unstructured text and (semi-)structured tables.

Many tables within documents and web pages are designed for direct human consumption and often lack the strict formatting that is expected in relational tables. This discrepancy poses significant challenges when querying them using languages such as Structured Query Language (SQL), integrating them with relational databases, and processing them within applications.

Large Language Models (LLMs) [3] have emerged as powerful tools for semantic parsing both textual and tabular data and performing complex tasks such as code generation. Trained on vast amount of Internet data, including both text and tables,

Table: Figure skating at the Asian Winter Games

Rank	Nation	Gold	Silver	Bronze	Total
1	China	13	9	13	35
2	Japan	7	10	7	24
3	Uzbekistan	1	2	3	6
4	Kazakhstan	2	2	0	4
5	North Korea	1	0	1	2
6	South Korea	0	0	2	2
Total		24	23	26	73

Q: who received more bronze medals: japan or south korea?
A: **Japan**

Figure 1.1: An example of table-based question answering.

and employing techniques such as Chain of Thought (CoT) prompting [4] and self-consistency [5], these models outperform many traditional models on various table reasoning tasks [6–9].

One of the key challenges in tabular data understanding tasks is handling large tables. While LLMs excel at understanding textual information efficiently, they operate within a limited token boundary. A significant challenge arises when utilizing LLMs for table reasoning tasks: if the LLM's token limit is exceeded, the model cannot effectively reason over large tables and may truncate the input, leading to incorrect results [10]. Conversely, if LLMs support a larger context window, there is a risk of hallucination where the model generates incorrect reasoning and outputs inaccurate results [10–12]. This underscores the delicate balance required in leveraging LLMs for reasoning tasks involving large, complex datasets.

At the same time, their performance in tasks involving tabular data, particularly those requiring symbolic reasoning (i.e. applying Structured Query Language (SQL) or python program), is often hindered by the structural variability and inconsistencies

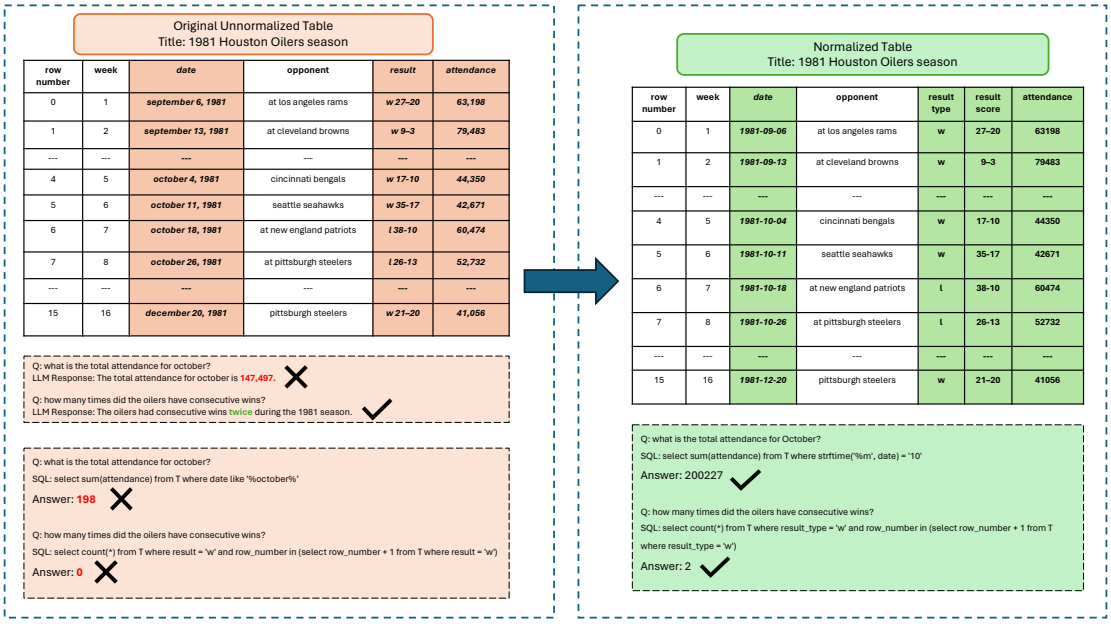

Figure 1.2: An example of a Table QA task, with the original unnormalized web table shown on the left and its normalized version on the right. Retrieve answers using a symbolic approach from the unnormalized table poses difficulties due to inconsistent formatting of ***date***, ***result*** and ***attendance*** columns. Also, direct querying with LLMs often fails for questions involving numerical operations. Normalization enables effective Text-to-SQL conversion, as shown by the normalized table on the right.

commonly found in web tables. Symbolic reasoning over tables necessitates a clear parsing of the table structure and values, and may involve constraining rows and columns, which can be challenging when dealing with unstructured or noisy web tables [13–16].

Consider the table QA task shown in Figure 1.2. Retrieving answers from the table on the left using a symbolic approach such as SQL is challenging due to the irregular structure of the data and the limitations of SQL. While an LLM may handle simple look-up questions, it struggles with tasks requiring complex aggregation and arithmetic operations. However, the normalized version of the same table, shown on the right, can be easily analyzed, allowing Text-to-SQL approaches to effectively obtain the answers to questions.

1.2 Thesis Objectives

Recent studies highlight the impressive capability of LLMs in reasoning over both text and tabular data. However, these works typically utilize the full table as context for reasoning [10–12, 15], limiting their ability to large tables. In particular, LLMs operate under a maximum token limit, and when processing a large table, there is a risk of potential truncation of the input or hallucination in the output [10, 11]. This limitation poses difficulties in handling large tables, making it impractical to encompass the entire table within the maximum token boundary of a prompt. Chen et al. (2023) [10] highlights that LLMs struggle to generalize when confronted with "large" tables containing 30 or more rows, leading to a decline in accuracy as the table size increases. While there have been works to decompose both questions and tables using LLMs [12], this line of work still requires providing the full table to the LLM and cannot scale to large tables. The question studied in the first part of this thesis is if the size of a table can be reduced before passing it to the language model without impacting its performance.

Existing models for table reasoning typically rely on a multi-step framework, where an LLM performs a sequence of actions such as adding columns before additional scripts are invoked to process data, retrieve cell values, or compute answers to questions [16–18]. These models are often dependent on question and table structure and do not address the root cause of table irregularity, making them less scalable. However, SQL is a powerful and proven tool for analyzing and querying large tables, unaffected by table size. SQL can only be applied effectively when table data are in a regular, normalized form (e.g., a relational database table). When attempting to apply symbolic reasoning, such as SQL operations, to irregular and unnormalized web tables, constructing appropriate conditions becomes challenging.

In the second part of the thesis, we investigate how we can leverage LLMs' textual understanding to effectively clean and normalize web tables. We explore how web

table normalization can enhance table reasoning tasks, particularly within the context of LLM-based symbolic reasoning. Our study focuses on table normalization as a stand-alone, one-time preprocessing step using LLMs to support symbolic reasoning on tabular data.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 discusses the related work. Chapter 3 details our method and experiments utilizing the Text-to-SQL approach to decompose tables and enhance table reasoning. Chapter 4 presents our method and evaluation of the tabular data normalization approach that enhances symbolic reasoning on tabular data. Chapter 5 concludes the thesis and proposes future research directions.

Chapter 2

Related Work

Our work is closely related to the literature on semantic parsing of questions and table schema (text-to-data), as well as reasoning applied to semi-structured tables (data-to-text). Additionally, it intersects with research on data wrangling and imputation.

2.1 Semantic Parsing: Text to Data

Table-based reasoning conventionally involves semantic parsing of questions and subsequent execution of the generated queries on tables. Traditional models in this domain were often domain-specific, supporting controlled natural language [19, 20], and posed challenges in adaptation to new domains or datasets. However, recent models leveraging machine learning techniques or large language models are trained on extensive datasets and query repositories, supporting a shift towards greater domain-independence. In particular, LLMs, when used with few-shot prompting, serve as powerful code generators, and techniques such as controlled decoding further improves the reliability of code generation [3, 14, 21–23].

Cross-domain benchmarks such as WikiSQL [24], Spider [25], CoSQL [26], SParC [27], and BIRD [28] have played a pivotal role in advancing this field, offering diverse examples of natural language queries paired with formal query language counterparts, such as SQL. These benchmarks serve as critical resources for testing the robustness and generalization capabilities of text-to-SQL models, driving improvements and in-

novation in this area of research.

Recent model LEVER [14] improved the code generation capabilities of LLMs trained on code (CodeLMs) by incorporating a mechanism to verify and re-rank CodeLM-generated programs based on their execution results. The combination of LEVER and Codex (code-davinci-002) achieves impressive results across the domains of table QA, math QA and basic Python programming. Glass et al. [29] explore methods to capture both row and column semantics, improving the model's query comprehension. Inner Table Retrieval (ITR) [30] employs a similarity-based approach for locating sub-tables. This model can select the required rows and colums for a given question and a table. These approaches involve pre-training and fine-tuning, which heavily rely on specific datasets. This reliance makes them inapplicable without access to a corresponding training dataset, while the need for optimal hyperparameters further limits their generalization.

In this line of work, the reasoning is generally done on questions and table schemata, with the expectation that the data in a table strictly adheres to the table schema (e.g., all values in a column having the same data type).

2.2 Table Reasoning: Data to Text

The relevant models can be categorized into more traditional models and recent LLM-based models. Many early models undergo pre-training on both tables and text to acquire a joint representation, utilizing this representation for reasoning without relying on symbolic execution. Notably, TaPas [8] retrieves masked information from tables, TAPEX [31] employs the BART model to emulate an SQL executor, ReasTAP [32] instills reasoning skills via pre-training, TABERT [33] encodes a subset of table content most pertinent to the input, and PASTA [6] pre-trains language models to be cognizant of common table-based operations. All these models have contributed to the progress on table-based reasoning. Despite achieving commendable results through pre-training on substantial datasets, these models still necessitate fine-tuning

on task-specific datasets [10].

LLMs have become competitive models in many domains and tasks including table reasoning, with their reasoning capabilities covering math, common sense, and symbolic reasoning. This is often done using few-shot prompts without fine-tuning [3].

We briefly review some of these LLM-based models that are relevant to our work.

Prompting Strategies The Table-CoT Model [10] generates the final answer to a question by employing in-context learning and chain-of-thought prompting to table-based tasks. The BINDER [34] model generates programs in a programming language, extending its capabilities to solve commonsense problems. The DATER [12] approach uses LLMs to decompose tables and questions for solving table-based QA and fact verification tasks. These approaches often excel using few-shot prompts without requiring fine-tuning. Their reasoning abilities can be further enhanced by breaking complex tasks into steps, employing methods like chain-of-thought (CoT) [4] prompting and Zero-CoT [35]. For instance, the Table-CoT [10] model utilizes in-context learning and CoT prompting to generate answers for table-based tasks. The reasoning capabilities can be further improved by carefully selecting examples in the prompt [36]

Supervised Fine-Tuning Several studies have utilized instruction tuning and supervised fine-tuning to enhance the performance of LLMs on table reasoning tasks. Notable examples include TableLLaMA [37] and TableGPT [38], which have shown significant improvements in specific applications. The Row-Column Intersection (RCI) model [29] utilizes training strategies to identify the relevant rows and columns of a table based on a question, thereby narrowing down the table size. However, it requires extensive training on a specific dataset, which fine-tunes the model to handle particular table structures and queries. These diverse approaches underscore the potential of LLMs in handling complex reasoning tasks involving tabular data.

Reasoning and Acting with LLM This line of work involves acting with an LLM agent back and forth using multitple loop. For instance, ReAcTable [16] adopts the

ReAct paradigm [39], encompassing step-by-step reasoning, code execution through external tools, intermediate table generation, and a majority voting mechanism. This method leverages LLMs to decompose the problem into multiple steps, each consisting of logical operations in the form of code to process tabular data as required. In a more recent model, LEVER [14] presents a method to enhance language-to-code generation by training to validate the generated programs based on their execution results.

Iterative Reading and Reasoning StructGPT [40] enhances the reasoning capabilities of large language models (LLMs) for structured data through an Iterative Reading-then-Reasoning approach. Despite its advantages, StructGPT's complexity and operational cost are increased by the necessity of passing entire tables to the LLM during the reading phase. This approach is constrained by token limits, which restricts the model's scalability when handling large tables.

In contrast, Chain-of-Table [18] extends the Chain-of-Thought methodology to the domain of tables, aiming to improve accuracy by transforming input tables and providing guidance to the LLM through intermediate tables during the reasoning process. However, this approach involves multiple intermediate steps and requires several calls to the LLM, which may impact efficiency.

Table reasoning approaches generally assume that the tables being analyzed are small enough to be directly input into the model. However, when dealing with large tables, it becomes challenging to fit the entire table within the prompt. This specific issue is one of the focuses of our investigation in this work, where we extract the relevant parts of the table to answer a question and reduce the table size.

Transforming Tables Chain-of-Table [18] enhances reasoning on tabular data by iteratively transforming and evolving table structures through a series of reasoning steps, including row/column selection, cell splitting to refine table representations for specific reasoning tasks. Their method employs in-context learning to direct LLMs in iteratively generating operations and updating the table, thus forming a chain of reasoning specific to tabular data. Liu et al. (2023) [17] explore the capabilities

of LLMs in interpreting and reasoning over tabular data, emphasizing robustness to structural perturbations, comparing textual and symbolic reasoning, and examining the potential of aggregating multiple reasoning pathways. Their findings indicate that structural variations in tables presenting the same content can significantly degrade performance, particularly in symbolic reasoning tasks. They propose a method for table structure normalization through transposition to mitigate this issue and find that while textual reasoning slightly outperforms symbolic reasoning, each approach has distinct strengths depending on the task.

2.3 Data wrangling and imputation

Normalizing tables is a crucial aspect of the broader data wrangling process, which involves processing, cleaning, and organizing data into a format suitable for further analysis. Considerable research has focused on data wrangling, addressing challenges such as error detection, data imputation, and standardization of data formats [41–43]. Recent approaches have leveraged the capabilities of LLMs for these tasks. For instance, [44] demonstrated the effectiveness of LLMs in identifying errors and imputing missing data, showcasing how these models can enhance the data wrangling process. By integrating LLMs, the efficiency and accuracy of preparing data for analysis can be significantly improved, streamlining and automating many aspects of data wrangling. Operations like normalizing numbers and dates can be incorporated into data processing workflows to aid in subsequent analysis [1].

Several studies leverage symbolic reasoning through Text-to-SQL or Python code for table-based reasoning tasks. However, for effectively utilizing a symbolic code generation approach with LLMs for table reasoning tasks, it is crucial to ensure that the table is in the proper format [1, 13–15, 21].

All these works highlight the importance of table normalization in improving LLMs' performance on tabular data, paving the way for more effective and accurate table reasoning models.

Chapter 3

Table Decomposition

This chapter explores our approach to enhancing table reasoning through table decomposition. Our primary objective is to extract pertinent sections of a table tailored to tasks such as TableQA or table-based fact verification. We investigate the utilization of Text-to-SQL capabilities inherent in LLMs to effectively decompose tables and retrieve essential components necessary for accurate task performance. By leveraging these capabilities, we aim to streamline the process of extracting and utilizing key information from tables, thereby improving the overall efficiency and effectiveness of table reasoning tasks.

3.1 Introduction

Tables represent the predominant form of structured information across various domains, encompassing databases, spreadsheets, open data repositories, web pages, and document collections. Creating effective natural language interfaces for tabular data presents a substantial challenge, particularly in accurately interpreting the semantics of table cells and discerning relationships between cell values in response to user queries.

Recent research underscores the impressive capabilities of LLMs in reasoning over both textual and tabular data. However, existing approaches typically rely on using the entire table as context for reasoning [10]. This reliance becomes problematic

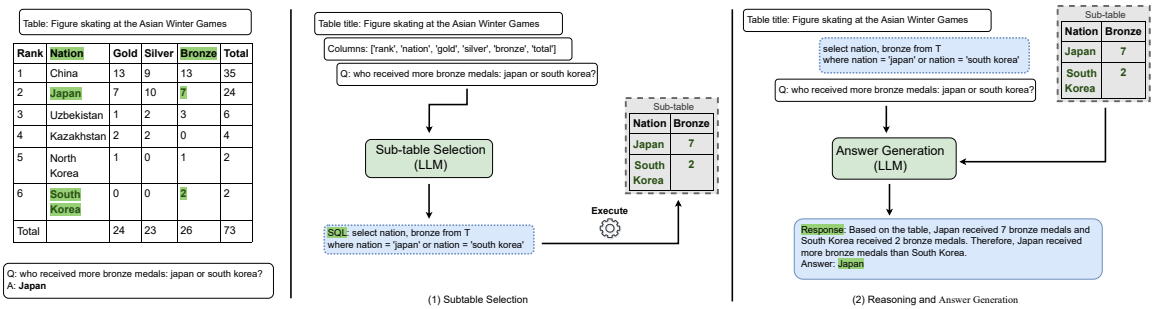

Figure 3.1: Overview of **TabSQLify**, consisting of two steps: **(1)** generating SQL queries from natural language questions or statements and executing the SQL queries on the original tables to obtain sub-tables containing only essential information, and **(2)** using LLMs with the sub-table and the question or claim to generate the answer.

when dealing with large tables, as LLMs are constrained by a maximum token limit. Processing such tables risks input truncation or generating erroneous outputs due to the complexity of handling extensive data within these constraints [10, 11]. This is a significant limitation, posing challenges in effectively utilizing LLMs for large-scale table reasoning tasks.

In this work, our aim is to leverage the symbolic representation capabilities of LLMs to reduce table size and their robustness to natural language variations for addressing formatting differences. Symbolic models, such as Text-to-SQL, are not affected by table size and can reliably scale to large tables. However, for reliable storage and querying in a relational database, tables are expected to adhere to a more rigorous formatting. Tables in the wild, such as those found on the web, often lack this formatting, necessitating substantial preprocessing and normalization efforts to convert the content [34]. LLMs are well-suited for resolving potential differences in the formating of rows and cell values. This work aims to strike a balance between table reasoning and table decomposition. Our approach involves using symbolic methods to narrow down the task to a targeted region in a table and then utilizes LLMs to reason over the limited relevant information.

We propose **TabSQLify**, a novel approach that integrates symbolic methods with the reasoning power of LLMs. TabSQLify leverages Text-to-SQL generation to de-

compose large tables into smaller and relevant sub-tables for table reasoning tasks. The method involves two key steps: **(1)** generating SQL queries from natural language questions or statements using LLMs under few-shot prompting, then executing the SQL queries on the original tables to obtain sub-tables containing only essential information for answering questions or verifying statements, and **(2)** using LLMs with the sub-table and the question or claim to generate the answer. The core concept of the approach is to utilize the natural language understanding and generation strengths of LLMs while reducing their burden in table encoding and reasoning (see Figure 3.1). Decomposing tables into sub-tables offers several advantages, including **(1)** reducing input length for improved scalability and efficiency in reasoning tasks involving large tables, **(2)** filtering out irrelevant and redundant information that do not contribute to the reasoning process, hence making the reasoning more focused, and **(3)** providing an intermediate representation (in this case, SQL queries and sub-tables) that is more interpretable and explainable for tracing and verification purposes.

We evaluate our method on four challenging table reasoning datasets: WikiTableQuestions [45], FeTaQA [46], TabFact [47] and WikiSQL [24]. Our evaluation on table-based question answering and fact verification tasks show that our method outperforms other LLM-based baselines, with gpt-3.5-turbo (chatgpt) as the LLM. Moreover, our method can significantly reduce the input length, making it more scalable and efficient for large-scale table reasoning applications than existing methods that require the full table context as input.

The contributions of our work are as follows:

1. We present a novel approach that utilizes Text-to-SQL generation to decompose tables into smaller, contextually relevant sub-tables, particularly designed for table reasoning tasks. This method offers a substantial reduction in table size, proving particularly advantageous for large tables that exceed the maximum allowable context window of LLMs.

2. Our model outperforms some of the leading models that employ multiple responses and self-consistency. Clearly using those techniques can further boost the performance of our method.

3. Our evaluation on challenging table reasoning datasets demonstrates the remarkable performance of our method compared to existing methods that rely on full tables as input. A comprehensive evaluation across various tasks is conducted to elucidate both the advantages and constraints of our approach.

3.2 Methods and Procedure

Our approach capitalizes on the proficiency of LLMs in parsing natural language text and generating SQL to enhance their capabilities in table reasoning. Large language models face challenges in accommodating extensive contextual information, especially when dealing with large tables that exceed their token limits. Increasing this size for large tables is unrealistic due to the quadratic time and memory complexities of self-attention mechanism in input length. Furthermore, LLMs are more likely to produce errors when handling lengthy contexts [10, 11]. To overcome these challenges, our work efficiently identifies and extracts relevant table parts, optimizing the input prompt size without sacrificing performance.

3.2.1 Table Preprocessing

Although tabular data is typically stored in a relational database and queried using SQL, many tables collected from web sources lack the rigorous structure and consistency that is needed for SQL queries to retrieve correct answers. It is generally a challenge to fully clean data from different sources or with no clean lineage records. Our hypothesis is that applying some general table cleaning and relaxing the granularity of retrievals to relevant rows and columns that have the answers, instead of the exact answers, makes the SQL engine more reliable. Of course, the exact answer

must be extracted at the end. This is done in our reasoning phase (§ 3.2.3) where an LLM is used, and it is better equipped to handle formatting differences.

For our table cleaning, we normalized numerical values and date fields. In particular, numerical values frequently feature commas, necessitating preprocessing to ensure consistency. To address this, we uniformly removed commas from all numerical entries. Additionally, the diverse date formats within the tables posed a challenge in generating accurate conditions for SQL queries. To address this, we standardized all date formats to the YYYY-MM-DD format. As an example, we converted numbers like "360,000" to "360000," and different date formats such as "31 October 2008," "31 Oct 2008" and "October 31, 2008" to the standardized "2008-10-31".

3.2.2 Subtable Selection

The subtable selection can be done by three strategies: **(1)** selecting essential columns, **(2)** selecting essential rows, and **(3)** selecting both essential columns and rows.

In this step, instead of feeding the entire table to the LLM, we provide essential table information such as the title, column names, and three example rows alongside the question. We utilize few-shot learning for this step, where we provide the LLM with a few examples. Subsequently, the LLM generates an SQL query to select the subtable based on this provided information. Selecting essential rows may require performing grouping and aggregation, and our generated SQL queries can include GROUP BY clauses and aggregation functions.

By selecting the essential columns and rows, we are reducing the context size while optimizing the relevance of information for subsequent reasoning tasks. When employing strategies (2) or (3), sometimes essential rows may not be safely extracted, for example returning an empty table due to noisy input. In those cases, we opt for the column selection strategy. The format of the prompt used for selecting necessary columns and row is described in Figure 3.2. In the subtable selection prompt, we provide the LLM with table metadata, including the table title, column names, and

three example rows. The LLM then generates an SQL query to extract the relevant subtable needed to answer the question. After receiving the SQL query, we execute it offline, outside of the LLM, to retrieve the necessary subtable.

```
Generate SQL for selecting the required rows and columns, given the question and table to answer the
question correctly.
 .... .... .... ....
SQLite table properties:

Table: 2012–13 Exeter City F.C. season(row_number,name,league,fa_cup,league_cup,jp_trophy,total)

3 example rows:
select * from T limit 3;
row_number | name | league | fa_cup | league_cup | jp_trophy | total
0 | scot bennett | 5 | 0 | 0 | 0 | 5
1 | danny coles | 3 | 0 | 0 | 0 | 3
2 | liam sercombe | 1 | 0 | 0 | 0 | 1

Q: does pat or john have the highest total?
SQL: select name, total from T where name like '%pat%' or name like '%john%'
.... .... .... ....
SQLite table properties:

Table: Figure skating at the Asian Winter Games (row_number, rank, nation, gold, silver, bronze, total)

3 example rows:
select * from T limit 3;
row_number | rank | nation | gold | silver | bronze | total
0 | 1 | china | 13 | 9 | 13 | 35
1 | 2 | japan | 7 | 10 | 7 | 24
2 | 3 | uzbekistan | 1 | 2 | 3 | 6

Q: who received more bronze medals: japan or south korea?
SQL:
----- ------ ------ ------
response:  select nation, bronze from T where nation = 'japan' or nation = 'south korea'
```

Figure 3.2: Prompt used for the subtable selection step of TabSQLify$_{\text{col+row}}$.

It is still conceivable that the output of the subtable selection step remains large, for example when finding the top-k most popular products for large values of k. We consider this limitation as inherent to the nature of the task and not specific to our approach since the sub-table containing all the items of the top-k is necessary to answer this question.

3.2.3 Reasoning and Answer Generation

In this step, an LLM is employed, wherein we input the SQL derived from the previous step, the subtable obtained by executing the SQL query and the question. Depending on the domain, additional contextual information, such as the surrounding text, may also be incorporated. This approach is adopted to help the model focus on the relevant parts for understanding the context and answering the question.

Based on the table title and execution result of the sql query bellow, find the answer to the given question correctly.
....
Table_title: Playa de Oro International Airport
SQL: select City, Passengers from T;

City | Passengers
United States, Los Angeles | 14,749
United States, Houston | 5,465
Canada, Calgary | 3,761
Canada, Saskatoon | 2,282
Canada, Vancouver | 2,103
United States, Phoenix | 1,829
Canada, Toronto | 1,202
Canada, Edmonton | 110
United States, Oakland | 107

Question: how many more passengers flew to los angeles than to saskatoon from manzanillo airport in 2013?
A: To find the answer to this question, let's think step by step. Based on the table, in 2013, the number of passengers who flew to Los Angeles from Manzanillo Airport was 14,749, while the number of passengers who flew to Saskatoon was 2,282. So, the difference in the number of passengers between Los Angeles and Saskatoon is 14,749 - 2,282 = 12,467. Therefore, the answer is 12,467.
Answer: 12,467
....
Table_title: Figure skating at the Asian Winter Games
SQL: select nation, bronze from T where nation = 'japan' or nation = 'south korea'

nation | bronze
japan | 7
south korea | 2

Question: who received more bronze medals: japan or south korea?
A: To find the answer to this question, let's think step by step.
----- ------ ------- -------
response: Based on the table, Japan received 7 bronze medals and South Korea received 2 bronze medals. Therefore, Japan received more bronze medals than South Korea.
Answer: Japan

Figure 3.3: Prompts used for the answer generation step.

We include the SQL query in the prompt to give the LLM additional context about the subtable. By knowing that the subtable is derived from executing the provided

SQL query, the LLM can improve its reasoning accuracy when generating an answer to the question. Moreover, we utilize few-shot learning techniques while adhering to the Chain-of-Thought prompting style. The format of the answer generation prompt is described in Figure 3.3.

3.3 Experimental Setup

3.3.1 Dataset

We assess our proposed approach across four datasets centered on reasoning with tables. Given our constraints on using LLMs, in terms of the number of requests and associated costs, our method is exclusively evaluated on the test sets of these datasets, with no fine-tuning on the training sets.

WikiTQ WikiTableQuestions (WikiTQ) contains complex questions annotated by crowd workers based on Wikipedia tables. These questions involve multiple complex operations, such as comparison, aggregation, and arithmetic operations, which require reasoning over multiple entries in a table. The standard test set contains 4,344 samples [45].

FetaQA Free-form Table Question Answering (FeTaQA) contains free-form table questions that require deep reasoning and understanding. These questions are usually hard because it requires processing information from different parts of the table. Unlike WikiTQ, this dataset annotates long free-form answers. Our approach is evaluated on the test set that contains 2,003 samples [46].

TabFact Table-Fact-Checking (TabFact) is a benchmark for verifying facts based on tables, which includes statements created by crowd workers using tables from Wikipedia. For example, a statement must be judged as either "True" or "False" based on the information in a given table. The accuracy is reported on the test-small set, which contains 2,024 statements and 298 tables [47].

WikiSQL WikiSQL is a simpler TableQA dataset, necessitating the filtering and

aggregation of information from the table content. Each question in WikiSQL is associated with a ground truth SQL query, from which we extract the gold answer and compare it with our results. We present the accuracy achieved on the test set of WikiSQL [24].

3.3.2 Implementation Details

In the experiments, we use gpt-3.5-turbo (chatgpt) as our language model. The prompt format mainly follows [23] and [48], which inputs the table schema and the first three table rows. We configured the in-context learning hyperparameters for gpt-3.5-turbo according to the specifications outlined in Table 3.1 and Table 3.2.

Sub table selection			
Parameter	**WikiTQ**	**FeTaQA**	**TabFact**
temperature	0.3	0.3	0.3
top_p	1	1	1
sample_n	1	1	1
max_tokens	100	100	100
num_shots	10	6	8

Table 3.1: Our hyper-parameter setting of LLM for selecting required column/row

Answer Generation			
Parameter	**WikiTQ**	**FeTaQA**	**TabFact**
temperature	0.7	0.7	0.6
top_p	1	1	1
sample_n	1	1	1
max_tokens	200	64	100
num_shots	2	6	4

Table 3.2: Our hyper-parameters setting of LLM for the answer generation

We set a lower temperature during the subtable selection step to ensure that the

SQL query generated by the LLM is more precise and focused, enabling it to extract the most relevant subtable for answering the question. Precision is crucial at this stage to avoid any ambiguity in selecting the necessary data. Conversely, we set a higher temperature during the answer generation step to encourage the LLM to explore a wider range of reasoning paths. This allows the model to consider diverse approaches and perspectives, ultimately leading to a more accurate and comprehensive final answer.

Our code and prompts are available at https://github.com/mahadi-nahid/TabSQLify.

3.3.3 Baselines

We compare our approach with several strong baseline methods. These methods can be split into two groups.

Pre-training and fine-tuning based models Our evaluation involves comparing our work with different models ranging from pre-training to fine-tuning. These models, pretrained on a large table corpus, aim to encode a given table as a plain sequence into an encoder and subsequently employ a decoder to generate an answer. As our baselines, we consider Table-BERT [47], LogicFactChecker [49], TaPas [8], SAT [50], TAPEX [31], GraPPa [51], PASTA [6] as our baslines. For FeTaQA evaluation, we compare our results against T5 [46, 52].

LLM based models For the LLM based methods with in-context learning, we compare against TableCoT [10], BINDER [34], DATER [12], StructGPT [40], ReAcTable [16], ITR [30], LEVER [14] and Chain-of-Table [18] as our baselines.

3.3.4 Evaluation metrics

For the WikiTQ and WikiSQL dataset, exact match (EM) accuracy was used to check if the predicted answers were the same as the correct ones. To account for different formatting of date and number fields, we added a pre-mactching check [34], consistent with preprocessing (§ 3.2.1). The accuracy of TabFact was determined using binary

classification accuracy. To evaluate FeTaQA, metrics such as ROUGE-1, ROUGE-2, and ROUGE-L [53] were used. However, ROUGE score lacks the ability to gauge the faithfulness and correctness of model-generated content. In line with Chen et al. (2023) [10], a human evaluation was conducted across four aspects: fluency (assessing linguistic errors), correctness (ensuring accurate answers to questions), faithfulness (verifying grounding on the input table), and adequacy (evaluating the comprehensiveness of the generated sentence in covering all answers) [46].

3.4 Results and Discussion

3.4.1 Model accuracy

As shown on Table 3.3, TabSQLify achieves an accuracy of 62.0% and 63.7% on the more challenging WikiTA dataset when reasoning is performed solely using the extracted columns and extracted rows, respectively. By extracting both the necessary columns and rows, we achieve an accuracy of 64.7%. Our model outperforms all pretrained models and LLM-based baselines, with chatgpt used as the LLM, on WikiTQ dataset [1]. It surpasses BINDER-Codex and achieves accuracy very close to the state-of-the-art model DATER. It is worth noting that, unlike our model, which considers only one response, both BINDER and DATER utilize 20 responses for the WikiTQ dataset to obtain the final answer.

For the TabFact dataset, as illustrated in Table 3.4, TabSQLify outperforms all LLM-based state-of-the-art approaches, with ChatGPT as the LLM. We achieve an accuracy of 79.5% when we extract the required sub-table by applying both column and row filtering. It is important to highlight that BINDER and DATER employ multiple responses and self-consistency to obtain the final answer. The reported results on TabFact are based on 50 responses for BINDER, 20 responses for DATER, and only one response for our model, hence it gives a lower bound of our model

[1]Codex was not available at the time of running our experiments, and the reported results are from the respective papers of our baselines.

Models	**Accuracy**
Agarwal et. al. 2019 [54]	44.1
Wang et. al. 2019 [9]	44.5
TaPas	48.8
GraPPa	52.7
LEVER	62.9
ITR	63.4
GPT-3 CoT	45.7
TableCoT-Codex	48.8
DATER-Codex	65.9
BINDER-Codex	61.9
ReAcTable-Codex	65.8
SQL-Codex	61.1
BINDER-chatgpt	55.4
DATER-chatgpt	52.8
ReAcTable-chatgpt	52.5
SQL-chatgpt	54.1
TableCoT-chatgpt	52.4
StructGPT	52.2
Chain-of-Table	59.9
$\text{TabSQLify}_{\text{col}}$	62.0
$\text{TabSQLify}_{\text{row}}$	63.7
$\text{TabSQLify}_{\text{col+row}}$	**64.7**

Table 3.3: Accuracy compared to the baselines on WikiTQ with the official evaluator.

performance.

Model	**Accuracy**
Table-BERT	68.1
LogicFactChecker	74.3
SAT	75.5
TaPas	83.9
TAPEX	85.9
SaMoE	86.7
PASTA	90.8
Human	92.1
TableCoT-Codex	72.6
DATER-Codex	85.6
BINDER-Codex	85.1
ReAcTable-Codex	83.1
ReAcTable-chatgpt	73.1
TableCoT-chatgpt	73.1
BINDER-chatgpt	79.1
DATER-chatgpt	78.0
Chain-of-Table	80.2
$\text{TabSQLify}_{\text{col}}$	77.0
$\text{TabSQLify}_{\text{row}}$	78.5
$\text{TabSQLify}_{\text{col+row}}$	**79.5**

Table 3.4: Experimental results on TabFact. Here, "Human" indicates the human performance [12]

For FeTaQA dataset, we achive a performance comparable to the baselines. As ROUGE metrics do not reflect the actual correctness of the model's responses, we manually evaluated 100 randomly chosen sample and quantified their performance in terms of fluency, correctness, adequacy and faithfulness. The performance is summarized in Tables 3.5 and 3.6. TabSQLify outperforms models based on fine-tuning

and pre-training, such as T5-large. The evaluation suggests that the model's output closely aligns with average human performance in terms of fluency, adequacy, and faithfulness. The correctness is notably impressive, although it falls behind human-level performance. This indicates that, utilizing TabSQLify results in high accuracy without the need for the entire table, showcasing the model's high level of precision in retrieving the relevant sub-table.

Model	**R-1**	**R-2**	**R-L**
T5-small	0.55	0.33	0.47
T5-base	0.61	0.39	0.51
T5-large	0.63	0.41	0.53
TableCoT-Codex	0.62	0.40	0.52
DATER-Codex	0.66	0.45	0.56
ReAcTable	0.71	0.46	0.61
TableCoT-chatgpt	0.62	0.39	0.51
$\text{TabSQLify}_{\text{col}}$	0.57	0.34	0.47
$\text{TabSQLify}_{\text{row}}$	0.60	0.37	0.49
$\text{TabSQLify}_{\text{col+row}}$	0.58	0.35	0.48

Table 3.5: Experimental results on FeTaQA.

Model	**Fluency**	**Correct**	**Adequate**	**Faithful**
T5-large	94.6	54.8	50.4	50.4
Human [10]	95	92.4	95.6	95.6
TableCoT-chatgpt	96	82	75	87
$\text{TabSQLify}_{\text{col}}$	98	83	79	85
$\text{TabSQLify}_{\text{row}}$	96	80	77	89
$\text{TabSQLify}_{\text{col+row}}$	97	88	84	93

Table 3.6: Human evaluation results on FeTaQA.

Apart from human evaluation, we analyze 100 sample outputs from the FeTaQA

Model	Precision	Recall	Relevancy	Faithfulness
TableCoT-chatgpt	0.44	0.94	0.94	0.73
$\text{TabSQLify}_{\text{col}}$	0.42	0.92	0.93	0.67
$\text{TabSQLify}_{\text{row}}$	0.45	0.97	0.94	0.73
$\text{TabSQLify}_{\text{col+row}}$	0.44	0.94	0.94	0.72

Table 3.7: RAGAS evaluation results on FeTaQA.

dataset using the RAGAS evaluator [55], a framework specifically designed for evaluating Retrieval Augmented Generation (RAG) pipelines. The RAGAS evaluation results is listed in Table 3.7. RAGAS assesses several key aspects: (1) Faithfulness: Evaluates the factual consistency of the answer concerning the context based on the question, (2) Context Precision: Measures the relevance of the retrieved context to the question, reflecting the quality of the retrieval pipeline, (3) Answer Relevancy: Assesses the relevance of the answer to the question, and (4) Context Recall: Measures the retriever's capability to retrieve all essential information required to answer the question. The performance of TabSQLify is comparable to that of Table-CoT-chatgpt, which utilized the full table context. Additionally, the RAGAS evaluation shows a similar trend to our human evaluation.

Model	Accuracy
SEQ2SQL	59.4%
StructGPT	65.6%
RCI [29]	89.8%
$\text{TabSQLify}_{\text{col+row}}$	**76.7%**

Table 3.8: Experimental results on WikiSQL. RCI is a fine tuning based model, and its results may not be directly comparable due to the model's high reliance on the training set.

TabSQLify shows an outstanding performance on the WikiSQL dataset, as demonstrated in Table 3.8. This dataset appears to be easier compared to the WikiTQ test

dataset, with our approach achieving 76.7% accuracy. Although the Row-Column Intersection (RCI) model [29] achieves higher accuracy, it benefits from extensive training on the WikiSQL dataset, which fine-tunes the model to handle specific table structures and queries. In contrast, our model achieves competitive performance without the need for any training on the dataset, demonstrating its versatility and effectiveness across different table reasoning tasks without relying on pre-existing training data. Morevever, In 70% of cases, our method can produce the answer in the first step, eliminating the need to pass the sub-table and question for the second step on WikiSQL dataset.

3.4.2 Scalability and robustness

We assessed the scalability and robustness of our model by imposing a token limit on each table across three datasets: WikiTQ, FeTaQA and TabFact. To accomplish this, we established cutoff thresholds to discard tokens exceeding these limits. Subsequently, we evaluated the model's performance within these constrained token boundaries. For the WikiTQ dataset, we set the cutoff threshold at 2000, while for both the TabFact and FeTaQA datasets, it was set to 600. Table 3.9 summarizes the distribution across different classes, illustrating the categories based on the percentage of discarded table tokens.

The cutoff percentage denotes the percentage of tokens that are truncated when the threshold is applied. For example, if a table has 4500 tokens and we set the threshold at 2000, then 2500 tokens of the original table are truncated, and the percentage is $2500/4500 = 55.56\%$. We separated the number of samples in different cutoff ranges and compared the results of those samples from different cutoff ranges in Tables 3.10, 3.11 and 3.12.

For the WikiTQ dataset, we set the threshold at 2000 tokens. In this case, out of 4,344 samples, there are 128 samples where more than 50% of the tokens of the original table are truncated if we want to pass the original table to the LLM with a

maximum token boundary of 2000. In our approach, TabSQLify selects the relevant limited subtable from the original table for a given question. This allows us to fit the subtable within the maximum token boundary when passing it to the LLM, resulting in improved performance. The aim of this experiment is to demonstrate that TabSQLify can be useful under limited token (context) boundary conditions.

Cut-off (%)	**WikiTQ**	**FeTaQA**	**TabFact**
0 - 10%	76	81	91
10 - 25%	89	143	141
25 - 50%	116	202	260
50% +	128	69	81

Table 3.9: The distribution of samples across various classes as a function of the percentage cut-off of table tokens.

Cut-off (%)	**TableCoT**	**TabSQLify$_{\text{col+row}}$**
0 - 10%	40.7	64.4
10 - 25%	49.4	60.6
25 - 50%	46.5	66.3
50% +	33.3	56.2

Table 3.10: Performance across different classes based on the percentage cut-off of table tokens in the WikiTQ dataset.

The evaluation results for the WikiTQ dataset are presented in Table 3.10. Our model consistently performs well within the specified token boundary. In contrast, the performance of TableCoT is subpar. We have observed a similar trend in the other two datasets (see Tables 3.11 and 3.12).

In the WikiTQ dataset, 128 tables contain more than 4000 tokens exceeding chat-GPT's maximum token limit (4096 tokens including table and question). Table 3.13 reports the performance on these instances. These results reveal that both BINDER [34] and DATER [12] face challenges when dealing with large tables. Specifically,

Cut-off (%)	TableCoT	TabSQLify$_{\text{col+row}}$
0 - 10%	76.9	79.1
10 - 25%	67.3	80.8
25 - 50%	63.0	70.0
50% +	55.5	72.8

Table 3.11: Performance across different classes based on the percentage cut-off of table tokens in the TabFact dataset

BINDER-Codex achieves only 29.6% accuracy, while DATER achieves an accuracy of 34.6%. BINDER-chatgpt fails to produce any correct answers for these large tables. On the other hand, Chain-of-Table [18] achieves an accuracy of 44.8%.

In contrast, our model outperforms these baselines significantly. It is crucial to note that our Table-CoT achieves this accuracy because the answers for questions about those large tables are typically in the upper part, fitting within the LLM's context boundary. If the answer is elsewhere, all models fail. On the other hand, our model has no issue with the answer's position in a table, making it scalable for large tables.

3.4.3 Table size reduction

Figure 3.4 demonstrates the average reduction in the number of table cells before and after employing TabSQLify$_{\text{col+row}}$ across three datasets. This reduction, from 183 to 32 cells in WikiTQ, indicates a substantial decrease in sub-table size while maintaining a strong performance. Likewise, similar trends can be observed in the TabFact, FetaQA and WikiSQL datasets. When utilizing both column and row filters to extract the required subtable, direct answers to questions may be found. Specifically, in the WikiTQ dataset, TabSQLify$_{\text{col+row}}$ successfully retrieves answers in 58% of cases by executing the generated query, requiring the answer generation step only for the remaining 42% of cases.

TableCoT			
Cut-off (%)	R-1	R-2	R-L
0 -10%	0.58	0.35	0.45
10-25%	0.60	0.37	0.50
25-50%	0.53	0.30	0.43
50% +	0.49	0.28	0.40
TabSQLify$_{\text{col+row}}$			
Cut-off (%)	R-1	R-2	R-L
0 -10%	0.62	0.39	0.50
10-25%	0.64	0.42	0.53
25-50%	0.55	0.32	0.44
50% +	0.51	0.31	0.41

Table 3.12: Performance across different classes based on the percentage cut-off of table tokens in the FeTaQA dataset

3.4.4 Error Analysis

An important advantage of TabSQLify is its ability to provide the intermediate stages of reasoning path, including SQL queries and sub-tables. To conduct our error analysis, we randomly selected 100 responses generated by TabSQLify$_{\text{row+col}}$ from the WikiTQ and TabFact test sets. The identified errors are categorized into incorrect columns, incorrect conditions, incorrect reasoning, and false negatives, as listed in Table 3.14.

In this context, a "missing column" refers to instances where TabSQLify either selects incorrect columns or omits necessary columns to answer the question. "missing rows" denotes situations where the generated SQL query contains an erroneous condition within the WHERE clause. Cases where the extracted sub-table is adequate to answer the question, but the LLM fails to provide a correct response, are labeled as "incorrect reasoning". Additionally, within the dataset, there are instances where the

Model	Acc (Large)
BINDER-Codex	29.6
BINDER-chatgpt	0.0
DATER-chatgpt	34.6
Table-CoT-chatgpt	35.1
Chain-of-Table [18]	44.8
$\text{TabSQLify}_{\text{col}}$	50.0
$\text{TabSQLify}_{\text{row}}$	**57.0**
$\text{TabSQLify}_{\text{col+row}}$	52.3

Table 3.13: Experimental results on Large (¿4000 tokens) tables from WikiTQ. As the input tables grow larger, we observe a decline in performance for strong baseline models.

Error Type	WikiTQ	TabFact
Missing Columns	6%	10%
Missing Rows	56%	32%
Incorrect Reasoning	29%	50%
Incorrect Annotation	9%	8%

Table 3.14: Error types of 100 samples from WikiTQ and TabFact of $\text{TabSQLify}_{\text{col+row}}$

gold answer is incorrect or misjudged by the evaluator, which we classify as "incorrect annotation".

From the table, we observe that out of 100 error cases from WikiTQ, 6% involve the generated SQL query missing columns, while 56% miss required rows. The irregular format of the text in the table is identified as the primary cause. Additionally, in 29% of cases, the reasoning is found to be incorrect, while 9% exhibit incorrect annotation. In the TabFact dataset, 10% of the time, the subtable selection query misses required columns, and in 32% of cases, it misses required rows. The main source of errors is incorrect reasoning, accounting for 50% of cases, while 8% involve incorrect annotations.

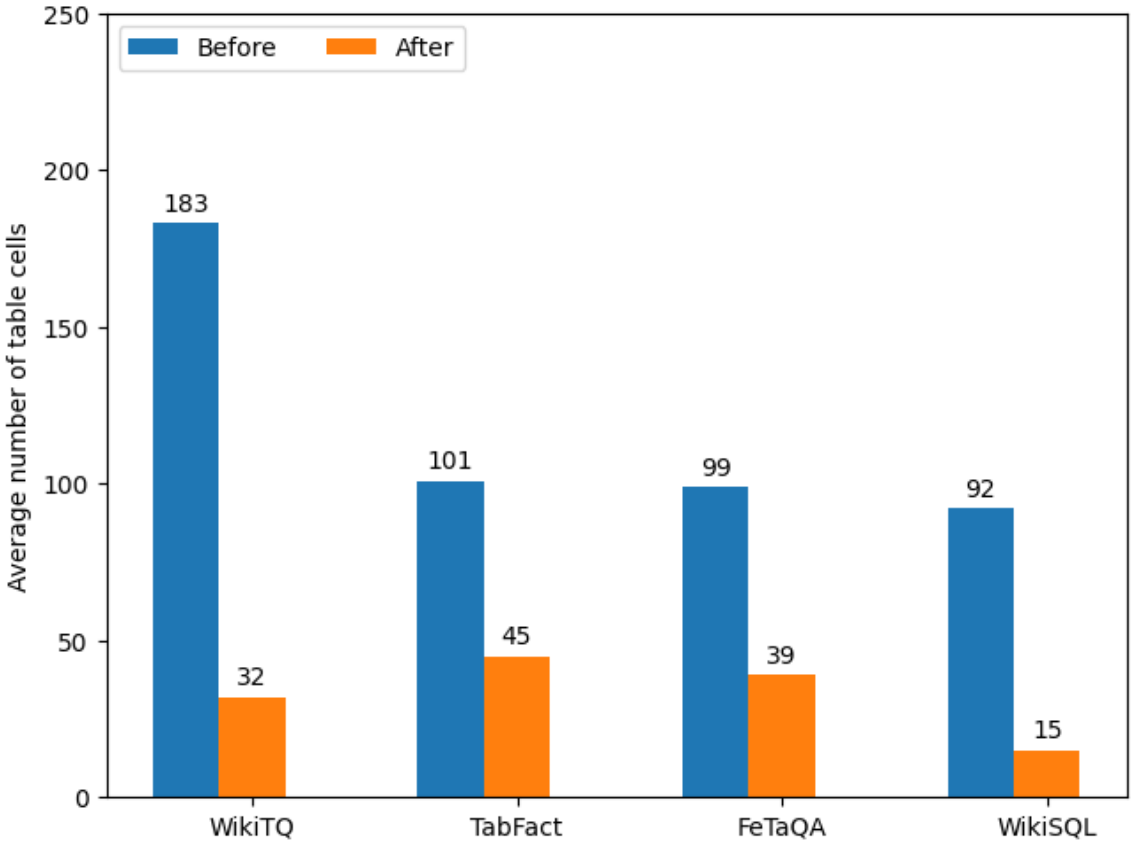

Figure 3.4: Reduction in table size using our row-col filtering across four datasets, showing a significant reduction of the table size.

3.4.5 Model Comparison

In this section, we conduct a comparative analysis of our model against two strong baselines, DATER [12] and BINDER [34]. DATER utilizes LLMs for decomposing both questions and tables. On the other hand, BINDER stands out by offering an Application Programming Interface (API) that extends language model (LM) functionalities to programming languages such as SQL and Python. This extension broadens its grammar coverage, enabling the model to address a more diverse range of questions. However, a drawback is that both DATER and BINDER necessitates sending the entire table to the LLM and face challenges when dealing with large tables. Both DATER and BINDER leverage self-consistency [5] strategies to bolster their performance, ensuring a higher level of consistency in their responses.

In our experiment we did not consider using self-consistence decoding strategy. Using self-consistency we can push the performance even higher. Our implementation does not require any additional processing on the SQL code, unlike BINDER, which necessitates a complex re-implementation of the SQL executor [14]. BINDER generates a total of 50 samples for a given table and question in the intermediate stages (Generate Neural-SQL: 50); while DATER generates 100 samples in its intermediate stages (table decomposition: 40; Generate Cloze: 20; Generate SQL: 20; reasoning: 20). In contrast, TabSQLify generates only two samples in total, making it simpler and more cost-effective. We summarize the comparison with DATER and BINDER in Table 3.15.

-	**DATER**	**BINDER**	**TabSQLify**
# stage	4	2	2
Max context size	8000	8000	4096
# of generated samples	100	50	2
sampling_n	20-50	20	1
Self Consistency	yes	yes	no
Table required	full	full	partial
Cost	high	high	low

Table 3.15: Comparison with the other LLM based models. TabSQLify is much simpler than the other approach.

Compared to the other LLM-based approach, our approach has several benefits: (1) Unlike other models our approach do not need to provide the table data to LLM to select the target portion of the table. Instead we utilize Text-to-SQL capability of LLMs. (2) Our approach requires partial table, not full table. (3) Our model can be applied in tight token boundary (4) Considering only one response our model can achive comparable performance while other top performing model uses more than 20 responses (5) Our approach is less costly and it requires less LLM calls which can be vital factor to reduce the cost.

3.5 Conclusions

Our proposed decomposition approach has shown promise across different table reasoning tasks, achieving remarkable performance compared to models that require the use of a full table. Our method is novel in leveraging Text-to-SQL generation to decompose tables into smaller and relevant sub-tables tailored for table understanding tasks. This approach provides a new perspective and direction for table reasoning research, and we hope it will inspire more future work on combining natural language understanding and structured data processing.

Chapter 4

Tabular Data Normalization

This chapter explores our approach to enhancing symbolic reasoning performance through the normalization of irregular web tables into standardized formats. The primary objective is to transform web tables with irregularities into normalized, structured tables. We leverage LLM to automate the normalization process, aiming to improve the consistency and reliability of tabular data for effective symbolic reasoning tasks.

By utilizing LLMs for web table normalization, we address challenges such as structural inconsistencies and varying data formats commonly found in web-based tables. This approach involves applying natural language understanding capabilities of LLMs to interpret and standardize diverse table structures, ensuring that data across different sources conform to a regular format suitable for robust symbolic reasoning.

Through experimental evaluation and analysis, we demonstrate the effectiveness of our normalization approach on enhancing LLM-based symbolic reasoning tasks. This research advances techniques for handling tabular data with LLMs, emphasizing the importance of data normalization for improving reasoning performance and efficiency in diverse domains.

4.1 Introduction

Tables are a fundamental format for structured data representation and are widely used across various sources, including relational databases, web pages, and financial documents. However, many tables within documents and web pages are designed for direct human consumption and often lack the strict formatting that is expected in relational tables. This discrepancy poses significant challenges when querying them using languages such as SQL, integrating them with relational databases, and processing them within applications.

In recent years, LLMs [3] have demonstrated remarkable performance in understanding tabular data [1, 12, 15–18, 21], outperforming many traditional models across various table reasoning tasks [6–9]. However, their performance in tasks involving tabular data, particularly those requiring symbolic reasoning, is often hindered by the structural variability and inconsistencies commonly found in web tables. Symbolic reasoning over tables necessitates a clear understanding of the table structure and values, and may involve constraining rows and columns, which can be challenging when dealing with unstructured or noisy web tables [13–16]. Our hypothesis is that normalizing ill-formatted tables can address this challenge, enabling the execution of symbolic programs (such as SQL or Python) on the tables and making reasoning tasks involving comparison, aggregation, and mathematical calculations more manageable. Moreover, normalization may enhance the explainability by allowing the tracking of the intermediate steps in reasoning.

Existing models for table reasoning typically rely on a multi-step framework, where an LLM performs a sequence of actions such as adding columns before additional scripts are invoked to process data, retrieve cell values, or compute answers to questions [16–18]. These models are often dependent on question and table structure and do not address the root cause of table irregularity, making them less scalable.

An alternative is normalizing tables, often part of a larger process known as data

wrangling, which involves processing, cleaning and organizing data into a format that is suitable for further analysis. Significant progress has been made on data wrangling [41–43], with recent approaches employing LLMs for tasks such as error detection and data imputation [44]. Selected operations, such as normalizing numbers and dates, may also be introduced into data processing pipelines to facilitate further analysis [1]. To the best of our knowledge, our work is the first to study table normalization as an stand-alone one-time preprocessing step using LLMs.

In this work, we introduce **NormTab**, a framework designed to normalize web tables to align them with the structured format of relational database tables. NormTab addresses challenges such as structural variance, mixed data formats, and extraneous information, thereby facilitating accurate and efficient symbolic reasoning and query processing using LLMs. Our work explores two key research questions:

- **RQ1:** How can we leverage LLMs' textual understanding to effectively clean and normalize web tables?
- **RQ2:** How can web table normalization enhance table reasoning tasks, particularly in the context of LLM-based symbolic reasoning?

Our proposed solution leverages the advanced textual understanding capabilities of LLMs to independently process and normalize web tables, without relying on specific questions. By normalizing tables in this manner, we enable a robust foundation for any downstream task involving table reasoning. This approach allows for multiple questions to be asked from a single, normalized table, significantly enhancing reasoning and query capabilities. Moreover, our normalization process only needs to be performed once, unlike other models that require repeated adjustments based on different questions, highlighting a key advantage of our approach.

Through a comprehensive experimental evaluation conducted on challenging web table datasets such as WikiTableQuestions [45] and TabFact [7], we assess the effectiveness of NormTab in improving table reasoning performance. These datasets

provide diverse examples of table structures and content, allowing us to thoroughly investigate the impact of web table normalization on LLM-based symbolic reasoning tasks. By addressing RQ1 and RQ2, we aim to demonstrate the importance of web table normalization and its potential to enhance the capabilities of LLMs in handling tabular data for complex reasoning tasks.

Key Contributions of our work are:

- We introduce NormTab, a novel framework that enhances LLMs' symbolic reasoning on tabular data by normalizing web tables. NormTab includes structure normalization (e.g., transposing tables, flattening rows and columns) and value normalization (e.g., removing extraneous strings, standardizing the formatting of dates and numbers) to ensure consistency and accuracy in reasoning tasks.

- We demonstrate how LLMs' textual understanding can be effectively utilized for data cleaning and transformation tasks, addressing challenges such as structural variance, mixed values, noise, and substring extraction in web tables

- We conduct extensive experimental evaluations using challenging web table datasets, including WikiTableQuestion and TabFact, to assess the effectiveness of NormTab in improving table reasoning performance, particularly in the context of LLM-based symbolic reasoning tasks.

4.2 Methods and Procedure

Our methodology encompasses several essential parts designed to ready web tables for proficient reasoning by LLMs.

4.2.1 Normalization Operations

The normalization operations in **NormTab** can be divided into two groups: **(1)** value normalization and **(2)** structural normalization. The former involves splitting cells

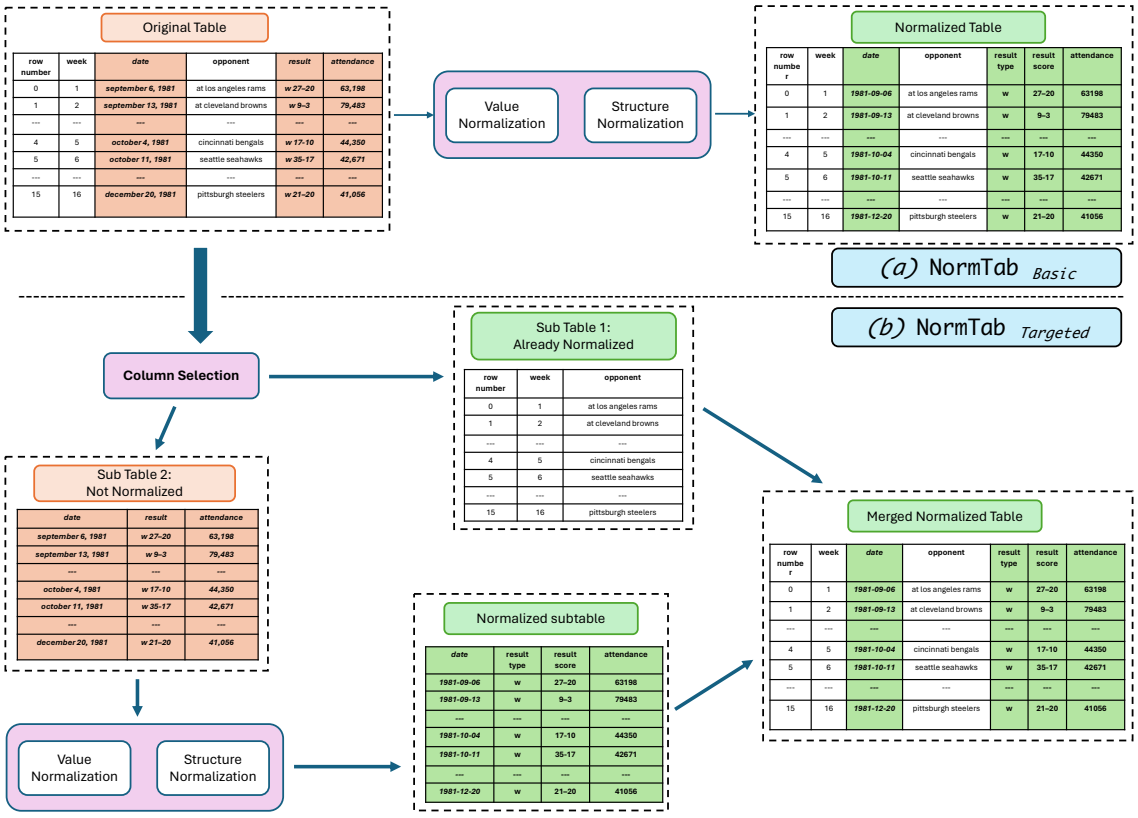

Figure 4.1: Overview of ***NormTab***. The methodology encompasses two distinct strategies: ***(a) Entire Table Normalization (NormTab$_{Basic}$):*** we provide the LLM with the entire web table along with specific instructions for cleaning and normalizing. The LLM reads the table and the instructions, then returns a cleaned and normalized version of the table. ***(b) Targeted Normalization (NormTab$_{Targeted}$):*** In this approach the LLM identifies and targets only the portions of the web table requiring normalization based on the table metadata and a few sample rows. The original table is split into two subtables: one for normalization and one already clean. The LLM processes the subtable that requires normalization then returned a cleaned version. Finally, the normalized subtable is merged with the clean portion, resulting in a fully cleaned and normalized table.

to add new columns, handling empty cells and value ranges, removing extraneous strings, and normalizing data formats such as dates and numerical values to ensure consistency and accuracy in reasoning tasks. Structural normalization, on the other hand, aims to detect structural variance by analyzing the first row and first column of a web table and determining whether a transposition is needed. If transposition is required, we address this issue by flipping the rows and columns.

Value Normalization: Our value normalization is based on the principle that every cell in a table must contain an atomic value (e.g., string, date, number), meaning that cell content cannot be composite or multi-valued. This principle, known as the first normal form in database systems [56], ensures that cell values can be smoothly queried and updated without introducing anomalies.

The process of value normalization involves several critical steps to ensure data consistency and accuracy. First, we focus on value splitting and extraction, identifying and splitting all composite columns. This may involve adding new columns as necessary while ensuring that no existing columns are deleted. Next, we standardize date and numerical values to a uniform format, paying special attention to any additional strings such as currency symbols, units or comma that may accompany numerical values. Additionally, we normalize all "N/A" and blank values to NULL to maintain consistency throughout the dataset. In SQL, Null values signify an attribute value that is not available or missing, and they are treated differently than any other values. SQL engines recognize the semantics of null values and consider this when processing queries. For columns containing value ranges, such as "2010/11" or "2015-2018", we split these into two separate columns to facilitate clearer data interpretation and processing.

An example of value normalization is shown in Figure 1.2. The original table presents date columns with dates in textual format, a result column combining match outcomes with scores, and an attendance column where numbers are written with commas. The value representation in the original table is more readable for humans;

however, this format poses challenges for symbolic programs to process. Our normalization process converts the date to the "YYYY-MM-DD" format and attendance values to a pure numerical format by removing commas. Additionally, NormTab splits the composite result column into two separate columns: "result_type" and "result_score", thereby organizing the data more effectively for analysis. This standardization is crucial for maintaining data integrity across the table.

Structural Normalization: Tables can be organized either row-oriented or column-oriented. In a row-oriented table, each row typically represents an entity or a relationship between entities, while each column describes an attribute of the entity or relationship. Column-oriented tables, on the other hand, are stored in a transposed fashion. Most traditional databases store data in a row-oriented format, which is well-supported across relational databases.

Our structure normalization primarily focuses on addressing structural differences between tables to enhance their usability for reasoning tasks. Initially, we carefully examine the table structure to determine if the first row resembles a header, indicating the table is row-oriented and requires no structural changes. However, if the first column appears to serve as the header, we transpose the table to normalize its structure, ensuring that the layout aligns with our adopted tabular format. Additionally, web tables sometimes include aggregated rows or columns, which can pose challenges if specific rows or columns need aggregation to answer a query. We handle these aggregated rows by disregarding any information present in the last row that pertains to aggregated data, such as "total", "sum", or "average". This step prevents redundant or misleading data from affecting subsequent analyses and ensures that the table remains clean and focused on the relevant data points.

4.2.2 Normalization Approach: NormTab

As depicted in Figure 4.1, our methodology for normalizing web tables involves two distinct approaches to leverage the capabilities of LLMs for enhancing symbolic rea-

soning and query capabilities.

Entire Table Normalization (NormTab-Basic): In the first approach, we provide the LLM with the entire table along with specific instructions for cleaning and normalizing. The LLM reads the table and the instructions, then returns a cleaned and normalized version of the table. However, we observed that many web tables contain portions already in a well-structured form, with only a few columns requiring normalization. To optimize this process, we developed a modified approach.

Targeted Normalization (NormTab-Targeted): To improve efficiency, we developed a modified approach that targets only the portions of the table requiring normalization. Our analysis of web tables revealed that often only a few columns need the normalization process. This realization led to a more optimized methodology. In this more refined approach, we first ask the LLM to identify which columns require normalization and cleaning, based on the table metadata (such as column headers and titles) and a few sample rows. Once these columns are identified, we split the original table into two subtables: one that requires normalization and cleaning, and one that is already normalized and clean. We then send only the subtable that needs normalization to the LLM along with the instructions. The LLM processes this subtable and returns a cleaned and normalized version. After normalization, we merge the normalized subtable with the already clean portion of the table. This approach not only improves the efficiency of the normalization task by reducing the amount of data sent to the LLM but also ensures that the resulting table is in a consistent and accurate format suitable for subsequent reasoning and querying tasks.

Following this, we analyze the overall structure of the merged table. With the assistance of the LLM, we determine whether the table needs to be transposed based on its layout. If needed, table transposition is performed outside of the LLM. Additionally, we check if the last row contains summarized or aggregated values and if so, NormTab ignore this row. This selective column normalization method reduces the workload on the LLM, enhances efficiency, and ensures that only the necessary parts

of the table are processed, thereby preserving the integrity of already structured data.

4.3 Experimental Setup

4.3.1 Dataset

We conduct experimental evaluations using two challenging web table datasets: WikiTableQuestion (WikiTQ)[45] and TabFact [7]. These datasets are specifically curated to test the reasoning capabilities of models on complex tabular data. WikiTQ comprises tables extracted from Wikipedia along with corresponding natural language questions, while TabFact consists of tables sourced from Wikipedia paired with textual facts. These datasets provide a diverse range of table structures and content, allowing us to thoroughly evaluate the performance of NormTab in enhancing table reasoning tasks.

The WikiTQ standard test set comprises 416 unique tables and 4,344 samples, while the TabFact standard test set includes 298 unique tables with 2,003 samples. By utilizing these datasets, we aim to demonstrate the effectiveness of web table normalization in improving the symbolic reasoning performance of LLMs, thereby highlighting the importance of addressing the challenges posed by web table irregularities.

4.3.2 Baselines and Evaluation Metrics

We compare our approach with several robust baseline methods, including TableCoT [10], BINDER [15], DATER [12], StructGPT [40], ReAcTable [16], Rethinking-Tab-Data [17], TabSQLify [1], and Chain-of-Table [18].

For the WikiTQ dataset, exact match (EM) accuracy was used to check if the predicted answers matched the correct ones. To address varying text formats, a pre-matching check using LLMs was incorporated [15]. The accuracy for TabFact was assessed using binary classification accuracy.

4.3.3 Implementation

We utilized gpt-3.5-turbo-0125 as the Language Model which supports 16k context window. We were inspired by the prompting style from [1, 17] in our implementation of NormTab. We configured the in-context learning hyperparameters for gpt-3.5-turbo-0125 according to the specifications outlined in Table 4.1.

Parameter	**Coll Selection**	**Transpose Detection**	**NormTab**
temperature	0.3	0.3	0.7
top_p	1	1	1
sample_n	1	1	1
max_tokens	100	100	4500
num_shots	6	1	1

Table 4.1: The hyper-parameters we set in NormTab

To compare performance, we employ few-shot in-context learning. This involves supplying the LLM with the table title, table header, question, and three example rows of the table, along with the question, to generate an SQL query. The SQL query is then executed on the table to obtain the answer. Details of the prompt and a sample output of NormTab can be found in Appendix A. [1].

4.4 Results and Discussion

In this section, we analyzed the performance of NormTab. To evaluate its impact, we conducted few-shot in-context learning experiments to generate SQL queries for answering specific questions. First, we performed experiments on unnormalized tables without any modifications. Then, we compared the performance on normalized tables. Additionally, we reported the performance of different normalization processes.

[1]The source code for the implementation discussed in this paper will be made publicly available upon acceptance of the paper.

4.4.1 Results on Downstream Tasks

Table 4.2 and Table 4.3 presents a comparison between the performance of NormTab and the other baselines on WikiTQ and TabFact datasets.

In the WikiTQ dataset, the results showed that after applying the targeted version of NormTab, we achieved 61.2% accuracy, surpassing the performance of other baseline models. The targeted NormTab approach performs slightly better than the basic version, where the entire table is passed to the LLMs. This suggests that LLMs may be more effective at normalization tasks when dealing with targeted smaller tables. Additionally, we gained about 10% improvement compared to the Text-to-SQL [21] model and SQL (gpt-3.5-turbo) model. Notably, Rethinking-Tab-Data [17] achieved an accuracy of 56.87% by addressing structural varience using LLMs and a Python agent. Chain-of-Table [18] employed an iterative sequence of operations to tailor complex tables to specific questions, achieving 59.94% accuracy. However, these and other baseline models are question-dependent. In contrast, our model adopts a straightforward and simple approach: it normalizes the table only once, irrespective of the question, enabling answers to be derived from the normalized table using program-aided symbolic reasoning.

In Table 4.3, we can observe a similar performance enhancement compared to the original table in table-based fact verification tasks. We achieved approximately a 6% performance improvement compared to the results of Text-to-SQL on the original table. It is worth noting that table-based fact verification differs from table-based question answering tasks. Generating a SQL query to verify a fact is more complex than simply retrieving an answer from the table. Although other models not employing program-aided symbolic reasoning perform better in this task, these models utilize LLMs for the verification task providing the whole table to the model. Our experimental results show promise for utilizing symbolic reasoning in such scenarios.

Model	**Acc (%)**
TableCoT [10]	52.40
Binder	56.74
Dater	52.80
ReAcTable	52.40
Rethinking-Tab-Data	56.87
Chain-of-Table	59.94
Text-to-SQL [21]	52.90
Text-to-SQL (gpt-3.5-turbo)	51.30
NormTab$_{Basic}$ + SQL (ours)	**60.80**
NormTab$_{Targeted}$ + SQL (ours)	**61.20**

Table 4.2: Performance comparison of NormTab on WikiTQ dataset. The results clearly demonstrate that NormTab significantly surpasses other models in accuracy when employing symbolic reasoning.

4.4.2 NormTab Evaluation

To assess the accuracy of various normalization operations, we evaluated the performance on 100 tables, with 50 tables from each dataset, WikiTQ and TabFact. Table 4.4 summarizes the accuracy of different normalization processes. NormTab demonstrated strong performance in normalizing dates and numbers, detecting transposition requirements, and handling aggregated summaries in the last row effectively. However, NormTab faced difficulties in extracting and cleaning values in certain critical tables where value extraction from the original table was particularly challenging. The column selection accuracy indicates that LLMs can be very effective in identifying columns where values are not in the proper format. However, the accuracy of splitting columns was low. Additional errors included managing value cleaning and handling "n/a" values. Although these tasks are challenging, the performance in these areas shows the potential for utilizing LLMs to address these tasks effectively.

NormTab has shown superior performance compared to several robust models,

Model	**Acc (%)**
TableCoT-chatgpt	73.10
Binder	79.17
Dater	78.01
Chain-of-Table	80.20
ReAcTable	73.10
Text-to-SQL [21]	64.71
Text-to-SQL (gpt-3.5-turbo)	62.32
NormTab$_{Basic}$ + SQL (ours)	**67.10**
NormTab$_{Targeted}$ + SQL (ours)	**68.90**

Table 4.3: Performance comparison of NormTab on TabFact dataset with other models.

demonstrating its efficacy in table normalization. A key advantage of NormTab is its use of program-aided symbolic reasoning, which streamlines code generation without requiring the entire table to be passed to the LLM. This enhances efficiency and eliminates dependencies on table size and answer position. With NormTab, only key elements like the title, header, and a few example rows are needed to generate SQL queries and obtain accurate answers. This approach reduces computational overhead while maintaining high accuracy, highlighting its practical utility in various table-based tasks.

4.4.3 Analysis

We conducted a detailed analysis of the impact of NormTab on the WikiTQ dataset. Table 4.5 shows that in **67%** of cases (Category A), performance improved after applying NormTab. In 24% of cases (Category B), performance remained unchanged, indicating no improvement. Additionally, in 9% of cases (Category C), performance actually decreased. The detailed experimental findings are summarized in Table 4.6.

Table 4.6 demonstrates that NormTab can improve overall performance by 9.9%.

Type	Description	Accuracy
Columns Selection	Selecting columns where values are not normalized	91.0%
Transpose Detection	Identify where transposition is required	97.0%
Last Row Aggregation	Ignoring aggregated last rows	100.0%
Split Column	Extract string and added to the new colums	87.0%
Date and Number	Standardize date and number format	100.0%
N/A value	Handling the table cells that contains N/A values	93.0%
Value Cleaning	Removing extraneous characters / strings	82.0%

Table 4.4: Accuracy of ***NormTab*** in various types of normalization.

Notably, in Category A, we observed a substantial enhancement of 16.27%. However, Categories B and C saw a slight decline in performance due to highly complex table values and structures.

The basic NormTab approach involves only one LLM call, but it requires passing the entire table to the language model as a prompt. This means the LLM must process a larger number of tokens, which can impact the overall normalization performance. Research indicates that more tokens can increase the likelihood of hallucination and can be more costly [1-3].

In contrast, the targeted NormTab approach first filters out the parts of the table that are already well-formatted, thereby reducing the number of tokens sent to the LLM by focusing only on the columns that need normalization. For example, consider a table with 15 rows and 8 columns, resulting in a total of 15 * 8 = 120 table cells. In the basic NormTab approach, all 120 cells are sent to the language model along with the instructions. However, if we identify that only 3 out of 8 columns require normalization, we only need to send 15 * 3 = 45 table cells. This reduction translates to 75 fewer table cells, which is a 62.5% reduction in table size. In Table 4.7 we summarize the table cell reduction of targeted NormTab on both datasets. We can see that we can reduce the table size by 72% in both datasets using targeted NormTab.

Although the targeted approach requires additional LLM calls for tasks such as

Categories	Description	% of Tables
A	Where performance enhanced after applying NormTab	**67%**
B	Where no change in performance after applying NormTab	24%
C	Where the performance decreased after applying NormTab	9%

Table 4.5: Categories of tables on WikiTQ test dataset.

-	Tables (A)	Tables (B,C)	Overall (A,B,C)
Original	46.28%	59.62%	51.30%
NormTab	**62.55%**	56.76%	**61.20%**
Change	**+16.27**	-2.86	**+9.9**

Table 4.6: Result breakdown on WikiTQ dataset.

column selection and transposition detection, the total number of tokens processed is still lower than in the basic NormTab approach. While the basic approach may be suitable for smaller tables, the reduction in table size is crucial for normalizing larger tables effectively. The target strategy involves multiple queries, but it refines the normalization process by concentrating on specific sub tasks, which may help reduce hallucination and errors.

While the performance improvement appears marginal, the significant token size reduction makes the targeted NormTab approach highly beneficial for larger tables. One challenge of implementing targeted normalization is the difficulty in effectively merging two subtables, particularly when the number of rows in the normalized subtable does not match the remaining subtable. This issue presents an opportunity for

Dataset	**NormTab-Basic**	**NormTab-Targeted**	**Reduction (%)**
WikiTQ (avg. # table cells)	152.26	41.82	72.53%
TabFact (avg. # table cells)	106.19	29.11	72.58%

Table 4.7: Efficiency of ***NormTab-Targeted***.

future work to develop more robust solutions.

4.5 Conclusions

In conclusion, our study introduces NormTab, a framework aimed at enhancing LLMs' performance on tabular data by normalizing web tables. Through our investigation, we have shown the significance of web table normalization in overcoming challenges such as mixed values and structural variance. By leveraging LLMs' textual understanding in data cleaning and normalization, NormTab improves table reasoning. Our experiments on challenging datasets demonstrate its effectiveness. Our work contributes to advancing techniques for LLMs in handling tabular data, emphasizing the importance of addressing web table challenges for improved performance. Further research can explore additional normalization strategies and extend NormTab's applicability across various domains. This would establish a robust foundation for a wide range of downstream tasks involving table reasoning.

Chapter 5

Conclusions & Future Work

5.1 Conclusions

Our proposed decomposition approach, TabSQLify has shown promise across different table reasoning tasks, achieving remarkable performance compared to models that require the use of a full table. This method is novel in leveraging Text-to-SQL generation to decompose tables into smaller and relevant sub-tables tailored for table reasoning tasks. This approach provides a new perspective and direction for table reasoning research, and we hope it will inspire more future work on combining natural language understanding and structured data processing.

In addition, our study presents NormTab, a framework designed to enhance the performance of LLMs on tabular data through the normalization of web tables. Our research highlights the importance of web table normalization in addressing issues such as mixed values and structural inconsistencies. By utilizing LLMs' capabilities in text comprehension for data cleaning and normalization, NormTab significantly improves table reasoning. The effectiveness of this approach is demonstrated through experiments on challenging datasets.

Overall, our work contributes to advancing techniques for LLMs in handling tabular data, emphasizing the importance of addressing challenges associated with web tables to enhance performance.

5.2 Future Work

Future research in table reasoning tasks using LLMs should focus on overcoming several challenges identified in our study. Key areas for exploration include enhancing LLMs' ability to handle larger and more complex tables by developing techniques that mitigate token limitations without compromising context integrity. Improving symbolic reasoning capabilities remains crucial, necessitating advancements in semantic parsing and logical inference to optimize query generation accuracy. Exploring multi-modal integration to combine textual, visual, and tabular data for comprehensive analysis represents an exciting frontier. Scaling and generalizing LLM-based approaches across diverse domains and datasets require robust models adaptable to varying data distributions and user needs. Investigating advanced data normalization methods to automate structural inconsistency detection and optimize data representations is essential for enhancing reliability and interpretability in LLM-driven reasoning tasks. Additionally, establishing standardized evaluation metrics and benchmarks tailored for LLM-based table reasoning will facilitate meaningful comparisons and advancements in the field. These avenues of research promise to extend the capabilities of LLMs in complex data analysis and decision-making contexts.

Limitation

Our TabSQLify approach is not without limitations. While it shows promise in reducing table size and maintaining strong performance, it may not be applicable for large tables where the size of the subtable can exceed the context window. Additionally, after preprocessing, the tables are stored in a relational format, but less regular tables may require additional preprocessing.

Despite the advancements brought by NormTab, there are several limitations. First, while our framework significantly enhances the symbolic reasoning capabilities of LLMs on tabular data, there remains room for improvement in the normal-

ization process, particularly with more complex table structures. Additionally, for larger tables, LLMs may sometimes produce hallucinated results, leading to inaccuracies in the normalized output, indicating a need for better handling of extensive datasets. Furthermore, when dealing with tables that contain extremely noisy data, LLMs struggle to effectively clean and normalize the information. As we measure the accuracy using the results obtained from LLM based Text-to-SQL model, it is important to note that some questions in the dataset may not directly map to SQL queries which may affect the performance.

Another limitation of this work is that we use LLMs from the GPT family. Future research could explore how different LLM architectures perform in comparison, offering opportunities for further investigation and enhancement.

Bibliography

[1] M. M. H. Nahid and D. Rafiei, "TabSQLify: Enhancing reasoning capabilities of LLMs through table decomposition," in *2024 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2024. [Online]. Available: https://aclanthology.org/2024.naacl-long.320.

[2] M. M. H. Nahid and D. Rafiei, "Normtab: Improving symbolic reasoning in llms through tabular data normalization," *arXiv preprint arXiv:2406.17961*, 2024.

[3] T. B. Brown *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[4] J. Wei *et al.*, "Chain of thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=_VjQlMeSB_J.

[5] X. Wang *et al.*, *Self-consistency improves chain of thought reasoning in language models*, 2023. arXiv: 2203.11171 [cs.CL].

[6] Z. Gu, J. Fan, N. Tang, P. Nakov, X. Zhao, and X. Du, "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*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds., Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 4971–4983. DOI: 10.18653/v1/2022.emnlp-main.331. [Online]. Available: https://aclanthology.org/2022.emnlp-main.331.

[7] W. Chen *et al.*, "Tabfact: A large-scale dataset for table-based fact verification," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rkeJRhNYDH.

[8] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. Eisenschlos, "TaPas: Weakly supervised table parsing via pre-training," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds., Online: Association for Computational Linguistics, Jul. 2020, pp. 4320–4333. DOI: 10.18653/v1/2020.acl-main.398. [Online]. Available: https://aclanthology.org/2020.acl-main.398.

[9] B. Wang, I. Titov, and M. Lapata, "Learning semantic parsers from denotations with latent structured alignments and abstract programs," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3774–3785. DOI: 10.18653/v1/D19-1391. [Online]. Available: https://aclanthology.org/D19-1391.

[10] W. Chen, "Large language models are few(1)-shot table reasoners," in *Findings of the Association for Computational Linguistics: EACL 2023*, A. Vlachos and I. Augenstein, Eds., Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 1120–1130. DOI: 10.18653/v1/2023.findings-eacl.83. [Online]. Available: https://aclanthology.org/2023.findings-eacl.83.

[11] Z. Ji *et al.*, "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, 2023, ISSN: 0360-0300. DOI: 10.1145/3571730. [Online]. Available: https://doi.org/10.1145/3571730.

[12] Y. Ye, B. Hui, M. Yang, B. Li, F. Huang, and Y. Li, "Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning," in *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '23, Taipei, Taiwan: Association for Computing Machinery, 2023, 174–184, ISBN: 9781450394086. DOI: 10.1145/3539618.3591708. [Online]. Available: https://doi.org/10.1145/3539618.3591708.

[13] M. Pourreza and D. Rafiei, "DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=p53QDxSIc5.

[14] A. Ni *et al.*, "Lever: Learning to verify language-to-code generation with execution," in *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, 2023.

[15] Z. Cheng *et al.*, "Binding language models in symbolic languages," in *The Eleventh International Conference on Learning Representations*, 2022.

[16] Y. Zhang, J. Henkel, A. Floratou, J. Cahoon, S. Deep, and J. M. Patel, *Reactable: Enhancing react for table question answering*, 2023. arXiv: 2310.00815 [cs.DB].

[17] T. Liu, F. Wang, and M. Chen, "Rethinking tabular data understanding with large language models," *arXiv preprint arXiv:2312.16702*, 2023.

[18] Z. Wang *et al.*, "Chain-of-table: Evolving tables in the reasoning chain for table understanding," *arXiv preprint arXiv:2401.04398*, 2024.

[19] A.-M. Popescu, O. Etzioni, and H. Kautz, "Towards a theory of natural language interfaces to databases," in *Proceedings of the 8th international conference on Intelligent user interfaces*, 2003, pp. 149–157.

[20] Y. Li, H. Yang, and H. Jagadish, "Nalix: A generic natural language search environment for xml data," *ACM Transactions on database systems (TODS)*, vol. 32, no. 4, 30–es, 2007.

[21] N. Rajkumar, R. Li, and D. Bahdanau, *Evaluating the text-to-sql capabilities of large language models*, 2022. arXiv: 2204.00498 [cs.CL].

[22] M. Pourreza and D. Rafiei, "Din-sql: Decomposed in-context learning of text-to-sql with self-correction," *arXiv preprint arXiv:2304.11015*, 2023.

[23] S. Chang and E. Fosler-Lussier, *How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings*, 2023. arXiv: 2305.11853 [cs.CL].

[24] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *CoRR*, vol. abs/1709.00103, 2017.

[25] T. Yu *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds., Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 3911–3921. DOI: 10.18653/v1/D18-1425. [Online]. Available: https://aclanthology.org/D18-1425.

[26] T. Yu *et al.*, "CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 1962–1979. DOI: 10.18653/v1/D19-1204. [Online]. Available: https://aclanthology.org/D19-1204.

[27] T. Yu *et al.*, "SParC: Cross-domain semantic parsing in context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds., Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 4511–4523. DOI: 10.18653/v1/P19-1443. [Online]. Available: https://aclanthology.org/P19-1443.

[28] J. Li *et al.*, *Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls*, 2023. arXiv: 2305.03111 [cs.CL].

[29] M. Glass *et al.*, "Capturing row and column semantics in transformer based question answering over tables," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, K. Toutanova *et al.*, Eds., Online: Association for Computational Linguistics, Jun. 2021, pp. 1212–1224. DOI: 10.18653/v1/2021.naacl-main.96. [Online]. Available: https://aclanthology.org/2021.naacl-main.96.

[30] W. Lin, R. Blloshmi, B. Byrne, A. de Gispert, and G. Iglesias, "An inner table retriever for robust table question answering," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 9909–9926. DOI: 10.18653/v1/2023.acl-long.551. [Online]. Available: https://aclanthology.org/2023.acl-long.551.

[31] Q. Liu *et al.*, *Tapex: Table pre-training via learning a neural sql executor*, 2022. arXiv: 2107.07653 [cs.CL].

[32] Y. Zhao, L. Nan, Z. Qi, R. Zhang, and D. Radev, "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*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds., Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 9006–9018. DOI: 10.18653/v1/2022.emnlp-main.615. [Online]. Available: https://aclanthology.org/2022.emnlp-main.615.

[33] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "TaBERT: Pretraining for joint understanding of textual and tabular data," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds., Online: Association for Computational Linguistics, Jul. 2020, pp. 8413–8426. DOI: 10.18653/v1/2020.acl-main.745. [Online]. Available: https://aclanthology.org/2020.acl-main.745.

[34] Z. Cheng *et al.*, *Binding language models in symbolic languages*, 2023. arXiv: 2210.02875 [cs.CL].

[35] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, *Large language models are zero-shot reasoners*, 2023. arXiv: 2205.11916 [cs.CL].

[36] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen, "What makes good in-context examples for GPT-3?" In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, E. Agirre, M. Apidianaki, and I. Vulić, Eds., Dublin, Ireland and Online: Association for Computational Linguistics, May 2022, pp. 100–114. DOI: 10.18653/v1/2022.deelio-1.10. [Online]. Available: https://aclanthology.org/2022.deelio-1.10.

[37] T. Zhang, X. Yue, Y. Li, and H. Sun, *Tablellama: Towards open large generalist models for tables*, 2023. arXiv: 2311.09206 [cs.CL].

[38] L. Zha *et al.*, "Tablegpt: Towards unifying tables, nature language and commands into one gpt," *arXiv preprint arXiv:2307.08674*, 2023.

[39] S. Yao *et al.*, *React: Synergizing reasoning and acting in language models*, 2023. arXiv: 2210.03629 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2210.03629.

[40] J. Jiang, K. Zhou, Z. Dong, K. Ye, X. Zhao, and J.-R. Wen, "StructGPT: A general framework for large language model to reason over structured data," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 9237–9251. DOI: 10.18653/v1/2023.emnlp-main.574. [Online]. Available: https://aclanthology.org/2023.emnlp-main.574.

[41] T. Furche, G. Gottlob, L. Libkin, G. Orsi, and N. W. Paton, "Data wrangling for big data: Challenges and opportunities," in *19th International Conference on Extending Database Technology*, 2016, pp. 473–478.

[42] Z. Abedjan *et al.*, "Detecting data errors: Where are we and what needs to be done?" *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 993–1004, 2016.

[43] T. Rattenbury, J. M. Hellerstein, J. Heer, S. Kandel, and C. Carreras, *Principles of data wrangling: Practical techniques for data preparation.* " O'Reilly Media, Inc.", 2017.

[44] A. Narayan, I. Chami, L. Orr, and C. Ré, "Can foundation models wrangle your data?" *Proceedings of the VLDB Endowment*, vol. 16, no. 4, pp. 738–746, 2022.

[45] P. Pasupat and P. Liang, "Compositional semantic parsing on semi-structured tables," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong and M. Strube, Eds., Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 1470–1480. DOI: 10.3115/v1/P15-1142. [Online]. Available: https://aclanthology.org/P15-1142.

[46] L. Nan *et al.*, "FeTaQA: Free-form table question answering," *Transactions of the Association for Computational Linguistics*, vol. 10, B. Roark and A. Nenkova, Eds., pp. 35–49, 2022. DOI: 10.1162/tacl_a_00446. [Online]. Available: https://aclanthology.org/2022.tacl-1.3.

[47] W. Chen *et al.*, *Tabfact: A large-scale dataset for table-based fact verification*, 2020. arXiv: 1909.02164 [cs.CL].

[48] C.-Y. Tai, Z. Chen, T. Zhang, X. Deng, and H. Sun, *Exploring chain-of-thought style prompting for text-to-sql*, 2023. arXiv: 2305.14215 [cs.CL].

[49] W. Zhong *et al.*, "LogicalFactChecker: Leveraging logical operations for fact checking with graph module network," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds., Online: Association for Computational Linguistics, Jul. 2020, pp. 6053–6065. DOI: 10.18653/v1/2020.acl-main.539. [Online]. Available: https://aclanthology.org/2020.acl-main.539.

[50] H. Zhang, Y. Wang, S. Wang, X. Cao, F. Zhang, and Z. Wang, "Table fact verification with structure-aware transformer," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds., Online: Association for Computational Linguistics, Nov. 2020, pp. 1624–1629. DOI: 10.18653/v1/2020.emnlp-main.126. [Online]. Available: https://aclanthology.org/2020.emnlp-main.126.

[51] T. Yu *et al.*, "Grappa: Grammar-augmented pre-training for table semantic parsing," in *International Conference on Learning Representations*, 2020.

[52] C. Raffel *et al.*, *Exploring the limits of transfer learning with a unified text-to-text transformer*, 2020. arXiv: 1910.10683 [cs.LG].

[53] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013.

[54] R. Agarwal, C. Liang, D. Schuurmans, and M. Norouzi, "Learning to generalize from sparse and underspecified rewards," in *International conference on machine learning*, PMLR, 2019, pp. 130–140.

[55] E. Gradients, *Ragas: Evaluation framework for retrieval augmented generation*, https://github.com/explodinggradients/ragas, 2023.

[56] M. Kifer, A. Bernstein, and P. M. Lewis, "Database systems: An application-oriented approach,"

[57] A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation and Compiling*. Englewood Cliffs, NJ: Prentice-Hall, 1972, vol. 1.

[58] American Psychological Association, *Publications Manual*. Washington, DC: American Psychological Association, 1983.

[59] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, "Alternation," *Journal of the Association for Computing Machinery*, vol. 28, no. 1, pp. 114–133, 1981. DOI: 10.1145/322234.322243.

[60] G. Andrew and J. Gao, "Scalable training of L1-regularized log-linear models," in *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 33–40.

[61] D. Gusfield, *Algorithms on Strings, Trees and Sequences*. Cambridge, UK: Cambridge University Press, 1997.

[62] M. S. Rasooli and J. R. Tetreault, "Yara parser: A fast and accurate dependency parser," *Computing Research Repository*, vol. arXiv:1503.06733, 2015, version 2. [Online]. Available: http://arxiv.org/abs/1503.06733.

[63] R. K. Ando and T. Zhang, "A framework for learning predictive structures from multiple tasks and unlabeled data," *Journal of Machine Learning Research*, vol. 6, pp. 1817–1853, Dec. 2005, ISSN: 1532-4435.

[64] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. [Online]. Available: https://aclanthology.org/P02-1040.

[65] J. Wei *et al.*, *Chain-of-thought prompting elicits reasoning in large language models*, 2023. arXiv: 2201.11903 [cs.CL].

[66] K. Kong *et al.*, "Opentab: Advancing large language models as open-domain table reasoners," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=Qa0ULgosc9.

[67] S. Patnaik, H. Changwal, M. Aggarwal, S. Bhatia, Y. Kumar, and B. Krishnamurthy, "CABINET: Content relevance-based noise reduction for table question answering," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=SQrHpTllXa.

[68] W. Zhou, M. Mesgar, H. Adel, and A. Friedrich, "Freb-tqa: A fine-grained robustness evaluation benchmark for table question answering," *arXiv preprint arXiv:2404.18585*, 2024.

[69] Y. Sui, M. Zhou, M. Zhou, S. Han, and D. Zhang, "Table meets llm: Can large language models understand structured table data? a benchmark and empirical study," in *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, ser. WSDM '24, ¡conf-loc¿, ¡city¿Merida¡/city¿, ¡country¿Mexico¡/country¿, ¡/conf-loc¿: Association for Computing Machinery, 2024, 645–654, ISBN: 9798400703713. DOI: 10.1145/3616855.3635752. [Online]. Available: https://doi.org/10.1145/3616855.3635752.

[70] P. Venetis *et al.*, "Recovering semantics of tables on the web," *Proc. VLDB Endow.*, vol. 4, no. 9, 528–538, 2011, ISSN: 2150-8097. DOI: 10.14778/2002938.2002939. [Online]. Available: https://doi.org/10.14778/2002938.2002939.

[71] Q. Liu *et al.*, "Tapex: Table pre-training via learning a neural sql executor," *arXiv preprint arXiv:2107.07653*, 2021.

[72] W. Zhong *et al.*, *Logicalfactchecker: Leveraging logical operations for fact checking with graph module network*, 2020. arXiv: 2004.13659.

Appendix A: NormTab Supplementary Contents

A.1 NormTab Output Example

Here is an example output of NormTab applied to a sample table from the WikiTQ dataset.

Sample Output of NormTab-Targeted

WikiTQ id: nu-129 title: 1981 Houston Oilers season

Table Columns: week, date, opponent, result, attendance

Original Table Markdown:

week	date	opponent	result	attendance
1	september 6, 1981	at los angeles rams	w 27-20	63,198
2	september 13, 1981	at cleveland browns	w 9-3	79,483
3	september 20, 1981	miami dolphins	l 16-10	47,379
4	september 27, 1981	at new york jets	l 33-17	50,309
5	october 4, 1981	cincinnati bengals	w 17-10	44,350
6	october 11, 1981	seattle seahawks	w 35-17	42,671
7	october 18, 1981	at new england patriots	l 38-10	60,474
8	october 26, 1981	at pittsburgh steelers	l 26-13	52,732
9	november 1, 1981	at cincinnati bengals	l 34-21	54,736
10	november 8, 1981	oakland raiders	w 17-16	45,519
11	november 15, 1981	at kansas city chiefs	l 23-10	73,984
12	november 22, 1981	new orleans saints	l 27-24	49,581
13	november 29, 1981	atlanta falcons	l 31-27	40,201
14	december 3, 1981	cleveland browns	w 17-13	44,502
15	december 13, 1981	at san francisco 49ers	l 28-6	55,707
16	december 20, 1981	pittsburgh steelers	w 21-20	41,056

Column Selection using LLM (Step 1): ['date', 'result', 'attendance']

Extracted Table (Normalization required):

date	result	attendance
september 6, 1981	w 27-20	63,198
september 13, 1981	w 9-3	79,483
september 20, 1981	l 16-10	47,379
september 27, 1981	l 33-17	50,309
october 4, 1981	w 17-10	44,350
october 11, 1981	w 35-17	42,671
october 18, 1981	l 38-10	60,474
october 26, 1981	l 26-13	52,732
november 1, 1981	l 34-21	54,736
november 8, 1981	w 17-16	45,519
november 15, 1981	l 23-10	73,984
november 22, 1981	l 27-24	49,581
november 29, 1981	l 31-27	40,201
december 3, 1981	w 17-13	44,502
december 13, 1981	l 28-6	55,707
december 20, 1981	w 21-20	41,056

Remaining subtable (Already Normalized):

week	opponent
1	at los angeles rams
2	at cleveland browns
3	miami dolphins
4	at new york jets
5	cincinnati bengals
6	seattle seahawks
7	at new england patriots
8	at pittsburgh steelers

9	at cincinnati bengals
10	oakland raiders
11	at kansas city chiefs
12	new orleans saints
13	atlanta falcons
14	cleveland browns
15	at san francisco 49ers
16	pittsburgh steelers

Normalized Sub-table (NormTab) Step 2:

date	result	result_score	attendance
1981-09-06	w	27-20	63198
1981-09-13	w	9-3	79483
1981-09-20	l	16-10	47379
1981-09-27	l	33-17	50309
1981-10-04	w	17-10	44350
1981-10-11	w	35-17	42671
1981-10-18	l	38-10	60474
1981-10-26	l	26-13	52732
1981-11-01	l	34-21	54736
1981-11-08	w	17-16	45519
1981-11-15	l	23-10	73984
1981-11-22	l	27-24	49581
1981-11-29	l	31-27	40201
1981-12-03	w	17-13	44502
1981-12-13	l	28-6	55707
1981-12-20	w	21-20	41056

Merged Table (Normalized) Step 2:

week	opponent	date	result	result_score	attendance
1	at los angeles rams	1981-09-06	w	27-20	63198
2	at cleveland browns	1981-09-13	w	9-3	79483
3	miami dolphins	1981-09-20	l	16-10	47379
4	at new york jets	1981-09-27	l	33-17	50309
5	cincinnati bengals	1981-10-04	w	17-10	44350
6	seattle seahawks	1981-10-11	w	35-17	42671
7	at new england patriots	1981-10-18	l	38-10	60474
8	at pittsburgh steelers	1981-10-26	l	26-13	52732
9	at cincinnati bengals	1981-11-01	l	34-21	54736
10	oakland raiders	1981-11-08	w	17-16	45519
11	at kansas city chiefs	1981-11-15	l	23-10	73984
12	new orleans saints	1981-11-22	l	27-24	49581
13	atlanta falcons	1981-11-29	l	31-27	40201
14	cleveland browns	1981-12-03	w	17-13	44502
15	at san francisco 49ers	1981-12-13	l	28-6	55707
16	pittsburgh steelers	1981-12-20	w	21-20	41056

****

Last row:

You are an advanced AI capable of analyzing and understanding information within tables. You are given the last row of a table. Your task is to detect if the last row has any information like aggregated rows such as ‘total’, ‘sum’ or 'average’.

Last row: ['16', 'pittsburgh steelers', '1981-12-20', 'w', '21-20', 41056]

Directly give your choice. Ensure the format is only "YES or NO" form, no other form, without any explanation.

response: NO

****

Transpose_promt:
You are an advanced AI capable of analyzing and understanding information within tables. Read the first 3 rows of the table regarding "1981 Houston Oilers season"

Table:
week	opponent	date	result	result_score	attendance
1	at los angeles rams	1981-09-06	w	27-20	63198
2	at cleveland browns	1981-09-13	w	9-3	79483
3	miami dolphins	1981-09-20	l	16-10	47379

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): (week, opponent, date, result, result_score, attendance)
(B): (week, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)

Directly give your choice. Ensure the format is only "(A) or (B)" form, no other form, without any explanation.
response: (A)
Need transposition: False

****
norm tab col: ['date', 'result', 'attendance']

Final merged norm_tab:
[['week', 'opponent', 'date', 'result', 'result_score', 'attendance'], ['1', 'at los angeles rams', '1981-09-06', 'w', '27-20', 63198], ['2', 'at cleveland browns', '1981-09-13', 'w', '9-3', 79483], ['3', 'miami dolphins', '1981-09-20', 'l', '16-10', 47379], ['4', 'at new york jets', '1981-09-27', 'l', '33-17', 50309], ['5', 'cincinnati bengals', '1981-10-04', 'w', '17-10', 44350], ['6', 'seattle seahawks', '1981-10-11', 'w', '35-17', 42671], ['7', 'at new england patriots', '1981-10-18', 'l', '38-10', 60474], ['8', 'at pittsburgh steelers', '1981-10-26', 'l', '26-13', 52732], ['9', 'at cincinnati bengals', '1981-11-01', 'l', '34-21', 54736], ['10', 'oakland raiders', '1981-11-08', 'w', '17-16', 45519], ['11', 'at kansas city chiefs', '1981-11-15', 'l', '23-10', 73984], ['12', 'new orleans saints', '1981-11-22', 'l', '27-24', 49581], ['13', 'atlanta falcons', '1981-11-29', 'l', '31-27', 40201], ['14', 'cleveland browns', '1981-12-03', 'w', '17-13', 44502], ['15', 'at san francisco 49ers', '1981-12-13', 'l', '28-6', 55707], ['16', 'pittsburgh steelers', '1981-12-20', 'w', '21-20', 41056]]

merged_norm_table_markdown:
week	opponent	date	result	result_score	attendance
1	at los angeles rams	1981-09-06	w	27-20	63198
2	at cleveland browns	1981-09-13	w	9-3	79483
3	miami dolphins	1981-09-20	l	16-10	47379
4	at new york jets	1981-09-27	l	33-17	50309
5	cincinnati bengals	1981-10-04	w	17-10	44350
6	seattle seahawks	1981-10-11	w	35-17	42671
7	at new england patriots	1981-10-18	l	38-10	60474
8	at pittsburgh steelers	1981-10-26	l	26-13	52732
9	at cincinnati bengals	1981-11-01	l	34-21	54736
10	oakland raiders	1981-11-08	w	17-16	45519
11	at kansas city chiefs	1981-11-15	l	23-10	73984
12	new orleans saints	1981-11-22	l	27-24	49581
13	atlanta falcons	1981-11-29	l	31-27	40201
14	cleveland browns	1981-12-03	w	17-13	44502
15	at san francisco 49ers	1981-12-13	l	28-6	55707
16	pittsburgh steelers	1981-12-20	w	21-20	41056

****

A.2 NormTab Prompt Templates

The prompt used in NormTab is described in the following Figures.

Column Selection Prompt

```
You are an advanced AI capable of analyzing and understanding information within tables.
Your task is to normalize a web table so that it can be converted as a relational database table.

### Instructions: Identify the columns based on the following instructions

1. Identify the columns If some of the values of a column needed to be extracted then extract the string and add it in new
columns.
2. Identify the columns that has date type value and the numerical value.
3. Identify the columns that has numerical values containing extra string such as '$' or units.
4. Identify the columns that has 'N/A',  blank or null.
5. Identify the columns that contain ranges  such as (20-2), 2010/11, 2015-2018 etc.

### Task: Your task is to identify which columns needed to be  normalized to convert this table as a regular normalized
relational database table so that we can run sqlite sql query over this table.

### Table:
Read the table below regarding "2008 Clásica de San Sebastián"

rank | cyclist | team | time | uci_protour_points
1 | alejandro valverde (esp) | caisse d'epargne | 5h 29' 10" | 40
2 | alexandr kolobnev (rus) | team csc saxo bank | s.t. | 30
3 | davide rebellin (ita) | gerolsteiner | s.t. | 25

Table Coll: (rank, cyclist, team, time, uci_protour_points)

### Response: normalize_coll = ['cyclist']

### Table:
Read the table below regarding "Sky Track Cycling"

date | competition | location | country | event | placing | rider | nationality
31 october 2008 | 2008–09 world cup | manchester | united kingdom | sprint | 1 | victoria pendleton | gbr
31 october 2008 | 2008–09 world cup | manchester | united kingdom | keirin | 2 | jason kenny | gbr
1 november 2008 | 2008–09 world cup | manchester | united kingdom | sprint | 1 | jason kenny | gbr

Table Coll: (date, competition, location, country, event, placing, rider, nationality)

### Response: normalize_coll = ['date', 'competition']
--- --- --- --- ---
--- --- --- --- ---
### Table:
Read the table below regarding "[TITLE]"

[3 ROWS OF THE TABLE]

### Output: Let's think step by step, and generate the final output based on the instructions without any explanation. Ensure
the final output is only "normalize_coll = [col1, col2, col3,…]"form, no other form.
### Response:
```

Figure A.1: Column Selection prompt.

Summarized Last Row Detection Prompt

You are an advanced AI capable of analyzing and understanding information within tables.

Task: You are given the last row of a table. Your task is to detect if the last row has any information like aggregated rows such as ‘total’, ‘sum’ or 'average’.

Last row: ***[LAST ROW AS A LIST]***

Directly give your choice. Ensure the format is only "YES or NO" form, no other form, without any explanation.

Response:

Transpose Detection Prompt

You are an advanced AI capable of analyzing and understanding information within tables.

Read the first 3 rows of the table regrading ***"[TITLE]"***

Table:

[3 ROWS OF THE 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 AS LIST]***

(B): ***[FIRST COLUMN AS LIST]***

Directly give your choice. Ensure the format is only "(A) or (B)" form, no other form, without any explanation.

Response:

Figure A.2: Summarized last row detection and transpose detection prompt.

NormTab Prompt

```
You are an advanced AI capable of analyzing and understanding information within tables.
Your task is to normalize a web table and convert it into a relational database table, enabling the execution of SQL queries on
the data.

Read the table below regarding "[TITLE]"

### Table:


[TABLE]


### Task: Your task is to normalize the structure and the values of each cell to convert this table as a regular normalized
relational database table so that we can run sqlite sql query over this table.


### Instructions:

1. If some of the values needed to be splitted or extracted then extract the string and add it in new columns. i.e. from
'alejandro valverde (esp)' country 'esp' can be extracted and added to the new column.
2. Make sure the date and the numerical value is normalized to a uniform format. The date format should be (YYYY-MM-DD).
3. Be cautious of numerical values that contain comma or any extra string such as '$', '%' or units.
4. Convert the 'N/A' or null values to blank.
5. Handle the columns that contain ranges such as 2010/11, 2015-2018 etc to  two separate columns.
6. Never delete any columns or rows.
7. Carefully remove extraneous characters if needed.



### Output: Let's think step by step and generate the final output table based on the instructions without any explanation.
Ensure the final output is only "normalized_table = [[col1, col2, col3,…], [row11, row 12, row13,.....],....]"form, no other
form.


### Response:
```

Figure A.3: NormTab Instruction prompt.

```
Generate SQL with no explanation given the question and table to answer the question correctly.

### SQLite table properties:

Table: Marek Plawgo(row_number,year,competition,venue,position,event,notes)

3 example rows:
 select * from T limit 3;
row_number | year | competition | venue | position | event | notes
0 | 1999 | european junior championships | riga, latvia | 4th | 400 m hurdles | 52.17
1 | 2000 | world junior championships | santiago, chile | 1st | 400 m hurdles | 49.23
2 | 2001 | world championships | edmonton, canada | 18th | 400 m hurdles | 49.8

Q: when was his first 1st place record?
SQL: select year from T where position = '1st' order by year asc limit 1

---- ----- ------ ----- ----
---- ----- ------ ----- ----
---- ----- ------ ----- ----

### SQLite table properties:

Table: Figure skating at the Asian Winter Games(row_number, rank, nation, gold, silver, bronze, total)

3 example rows:
 select * from T limit 3;
 row_number | rank | nation | gold | silver | bronze | total
0 | 1 | china | 13 | 9 | 13 | 35
1 | 2 | japan | 7 | 10 | 7 | 24
2 | 3 | uzbekistan | 1 | 2 | 3 | 6

Q: what is the average number of gold medals won by china, japan, and north korea?
SQL: select avg(gold) from T where nation in ('china', 'japan', 'north korea')

### SQLite table properties:

Table: [TITLE] ([COLUMN NAMES])

3 example rows:
 select * from T limit 3;

 [THREE EXAMPLE ROWS]

Q: [QUESTION]
SQL:
```

Text-to-SQL Prompt template (WikiTQ)

Figure A.4: Text-to-SQL prompt template for the Table QA Task on the WikiTQ dataset.

```
Generate SQL given the statement and table to verify the statement correctly.

### SQLite table properties:

Table: 1947 kentucky wildcats football team(row_number, game, date, opponent, result, wildcats_points, opponents, record)

3 example rows:
 select * from T limit 3;
row_number | game | date | opponent | result | wildcats_points | opponents | record
0 | 1 | sept 20 | ole miss | loss | 7 | 14 | 0 - 1
1 | 2 | sept 27 | cincinnati | win | 20 | 0 | 1 - 1
2 | 3 | oct 4 | xavier | win | 20 | 7 | 2 - 1

Q: the wildcats kept the opposing team scoreless in four games
SQL: SELECT (SELECT COUNT(*) FROM T WHERE opponents = 0) = 4

---- ----- ------ ----- ----
---- ----- ------ ----- ----
---- ----- ------ ----- ----

### SQLite table properties:

Table: katsuya inoue(row_number, res, record, opponent, method, event, round, time, location)

3 example rows:
 select * from T limit 3;
row_number | res | record | opponent | method | event | round | time | location
0 | loss | 19 - 9 - 4 | naoyuki kotani | submission (armbar) | pancrase - impressive tour 9 | 1 | 1:44 | tokyo , japan
1 | loss | 19 - 8 - 4 | kota okazawa | ko (punch) | pancrase - impressive tour 4 | 1 | 2:42 | tokyo , japan
2 | win | 19 - 7 - 4 | katsuhiko nagata | decision (unanimous) | gcm - cage force 17 | 3 | 5:0 | tokyo , japan

Q: in tokyo , japan , hikaru sato 's match ended before round 2
SQL: SELECT (SELECT COUNT(*) FROM T WHERE location = 'tokyo , japan' AND round < 2) > 0;

### SQLite table properties:

Table: [TITLE] ([COLUMN NAMES])

3 example rows:
 select * from T limit 3;

 [THREE EXAMPLE ROWS]

Q: [STATEMENT]
SQL:
```

Text-to-SQL Prompt template (TabFac)

Figure A.5: Text-to-SQL prompt template for the Table-based Fact Verification Task on the TabFact dataset.