Large Language Model for Table Processing: A Survey

Weizheng Lu[1] , **Jiaming Zhang**[1] , **Jing Zhang**[1*] and **Yueguo Chen**[1]
[1]Renmin University of China
{luweizheng,zhangjiaming2002,zhang-jing,chenyueguo}@ruc.edu.cn

Abstract

Tables, typically two-dimensional and structured to store large amounts of data, are essential in daily activities like database queries, spreadsheet calculations, and generating reports from web tables. Automating these table-centric tasks with Large Language Models (LLMs) offers significant public benefits, garnering interest from academia and industry. This survey provides an extensive overview of table tasks, encompassing not only the traditional areas like table question answering (Table QA) and fact verification, but also newly emphasized aspects such as table manipulation and advanced table data analysis. Additionally, it goes beyond the early strategies of pre-training and fine-tuning small language models, to include recent paradigms in LLM usage. The focus here is particularly on instruction-tuning, prompting, and agent-based approaches within the realm of LLMs. Finally, we highlight several challenges, ranging from private deployment and efficient inference to the development of extensive benchmarks for table manipulation and advanced data analysis.

1 Introduction

In this data-driven era, a substantial volume of data is structured and stored in the form of tables [Dong *et al.*, 2022]. Everyday tasks involving tables, such as database queries, spreadsheet calculations, or generating reports from web tables, are common in our daily lives. While some of these tasks are tedious and error-prone, others require specialized skills and should be simplified for broader accessibility. The automation of tasks related to tables offers significant benefits to the general public, and consequently, it has attracted considerable interest from both academic and industrial sectors [Badaro *et al.*, 2023].

Recently, large language models (LLMs) have showcased their effectiveness and flexibility in diverse tasks, marking significant advancements in AI research [Zhao *et al.*, 2023a]. This success in extensive natural language processing applications has inspired researchers to explore LLMs in table tasks. However, the nature of tables is quite different from the plain text commonly used during the pre-training of LLMs. Tables, unlike natural language corpora [Raffel *et al.*, 2020], have distinct characteristics.

- **Structured Data:** Tables are inherently structured, composed of rows and columns, each with its own schema that outlines the data's semantics and their interrelations. Humans can effortlessly interpret tables both vertically and horizontally, but LLMs, primarily trained with sequential text data, struggle with understanding the multidimensional aspects of tables.
- **Complex Reasoning:** Tasks in table processing often require numerical operations (like comparisons or aggregations), data preparation (such as column type annotation and missing value detection), and more sophisticated analyses (including feature engineering and visualization). These tasks demand intricate reasoning, the ability to decompose problems into multiple steps, and logical operations, thereby posing significant challenges to machine intelligence.
- **Utilizing External Tools:** In real-world scenarios, humans often depend on specialized tools such as Microsoft Excel, Python, or SQL for interacting with tables. For effective table processing, LLMs need to be adept at integrating and using these external tools.

The unique challenges presented by table processing tasks emphasize the need to tailor LLMs for these specific purposes. Early research, such as TaBERT [Yin *et al.*, 2020], TaPas [Herzig *et al.*, 2020], TURL [Deng *et al.*, 2020], and TaPEx [Liu *et al.*, 2022], adhere to the paradigm of pre-training or fine-tuning neural language models for tables. These methods adapt model architectures, including position embeddings, attention mechanisms, and learning objectives for pretraining tasks. While these approaches yield good results, they are largely confined to specific table tasks like table question answering (Table QA) and fact verification. Additionally, the BERT or BART models they utilize are not sufficiently large or versatile to handle a broader range of table tasks. Latest LLM-based approaches tackle table tasks in three primary ways: (1) curating an instruction-tuning dataset and fine-tuning a "table foundational model" that could handle various downstream tasks [Zhang *et al.*, 2023a; Li *et al.*, 2023d]; (2) prompting an LLM by utilizing the LLM's strong

*Corresponding author

reasoning ability to understand table data [Sui *et al.*, 2024]; and (3) building an LLM-powered agent that can call external tools [Hu *et al.*, 2024; Li *et al.*, 2023b]. These newer methods leverage LLM-specific technologies, such as instruction-tuning [Wei *et al.*, 2022a], in-context learning [Brown *et al.*, 2020], chain-of-thought reasoning [Wei *et al.*, 2022b], and the development of autonomous agents [Wang *et al.*, 2023a], showcasing a more versatile and comprehensive approach to table processing.

The main contribution of this survey is its extensive coverage of a wide range of table tasks, including recently proposed table manipulation and advanced data analysis. Additionally, it categorizes methods based on the latest paradigms in LLM usage, specifically focusing on instruction-tuning, prompting, and LLM-powered agent approaches. We collect resources such as papers, code, and datasets, which can be accessed at https://github.com/godaai/llm-table-survey.

Comparison with Related Surveys. Earlier surveys, such as those by Dong et al. [Dong *et al.*, 2022] and Badaro et al. [Badaro *et al.*, 2023], primarily concentrate on pre-training or fine-tuning techniques using smaller models like BERT [Herzig *et al.*, 2020; Yin *et al.*, 2020] or BART [Liu *et al.*, 2022]. However, they do not address methods based on LLMs, particularly those involving prompting strategies and agent-based approaches. Additionally, some surveys are confined to specific subsets of table tasks. For instance, Jin et al. [Jin *et al.*, 2022] focus solely on tableQA, while Qin et al. [Qin *et al.*, 2022] concentrate on text-to-SQL conversion, overlooking tasks such as table manipulation and advanced data analysis.

2 Table Tasks and Benchmarks

Tables are prevalent data structures that organize and manipulate knowledge and information in almost every domain. We briefly summarize table tasks and benchmarks.

2.1 Table Definition

In this paper, we mainly focus on structured tables, which are grids of cells arranged in rows and columns [Zhang and Balog, 2020]. Every column in a table signifies a specific attribute, each having its own data type (e.g., numeric, string, or date), and each row forms a record, populated with diverse attribute values. A cell is the intersection of a row and a column. Cells are basic units to store text, numerical values, formulas, etc [Dong *et al.*, 2022]. The two-dimensional structure endows the data with a schema, which includes column names, data types, constraints, and relevant relationships with other tables. Tables are mainly stored and presented in the following forms:

- **Spreadsheet** (SS). Spreadsheets are used by one-tenth of the global population [Rahman *et al.*, 2020]. Microsoft Excel is the most popular desktop-based spreadsheet software. Excel has its own file format (xlsx), comes equipped with built-in formulas, data visualization, and even allows table programming (Visual Basic for Applications, VBA).
- **Web Table** (WT). The Web hosts a multitude of tables, created for diverse purposes and containing valuable information. These tables exist in varied formats, ranging from HTML tables within webpages to markdown, JSON, and XML files. Web tables prove beneficial in applications such as column type annotation or entity linking [Zhang and Balog, 2020].
- **Database** (DB). Tables in databases are highly structured. Each database table is explicitly defined with a schema at the creation time. The associations between tables are defined through foreign keys [Li *et al.*, 2023a]. Compared to spreadsheets and Web tables, interaction with database tables necessitates using programming languages or SQL. Thus, database tables primarily target on individuals with programming experience.

2.2 Differences Between Table and Text

Many AI methodologies migrate text modeling techniques to tables. Therefore, we should consider the differences between tables and text. [Li *et al.*, 2023d] outlines the main distinctions between them. Texts are (1) one-directional; (2) typically read from left to right; and (3) the swapping of two tokens usually alters the sentence's meaning. On the other hand, tables are (1) two-dimensional, requiring both horizontal and vertical reading; (2) their understanding heavily relies on schemas or header names; and (3) they remain unaffected by row and column permutations.

2.3 Table Tasks

Classification

Table 1 presents our classification of table tasks. Table question answering and fact verification are the most traditional table tasks. We categorize them into a single group because they all involve extracting knowledge from tables to answer questions. Table-to-text produces a description based on table data. Table manipulation modifies or updates the table, such as adding or deleting rows, creating a new column by summing up other columns, etc. Table interpretation tasks aim to comprehend the structure and content of tables by inferring the semantics of columns and rows. Table augmentation involves populating rows, column names, or cells. Text-to-SQL converts NL questions into a machine-understandable logical form, namely SQLs. Advanced data analysis refers to high-level tasks requiring complex reasoning or creativity, such as feature engineering or data visualization.

End-users' Perspective

Researchers often focus solely on designing new methods to improve performance on benchmarks. However, end-users are primarily interested in how table-related AI systems can boost their productivity, rather than the specifics of benchmarks. End-user requirements vary based on their roles; common users typically need table querying and manipulation capabilities, while data engineers seek tools for tabular data preparation and advanced analysis.

Table Querying and Manipulation refers to the first three categories in Table 1 (i.e., *Table QA*, *Table-to-text*, *Manipulation*). This family involves querying tables, or generating text descriptions through NL, or managing tables via spreadsheets. Most users can perform these operations without programming skills. AI systems should enable individuals to process tables through NL questions or instructions.

Table 1: Table tasks, datasets, input and output. The abbreviations are SS: Spreadsheet, WT: Web Tables, and DB: Database.

Category	Representative Task	Dataset	Knowledge Input	User Input	Output
Table QA	Question answeing Fact verification	WikiTQ [Pasupat and Liang, 2015] WikiSQL [Zhong *et al.*, 2017] HybridQA [Chen *et al.*, 2020b] SQA [Iyyer *et al.*, 2017] FeTaQA [Nan *et al.*, 2022] TabFact [Chen *et al.*, 2020a]	WT	Question	Answer
Table-to-text	Summary Dialogue	ToTTo [Parikh *et al.*, 2020] DART [Nan *et al.*, 2021]	WT	Instruction	Text
Manipulation	Insert Delete Alter	SpreadsheetCoder [Chen *et al.*, 2021b] Sheetcopilot [Li *et al.*, 2023b]	SS	Instruction	Formula SS
Interpretation	Entity linking Column type annotation Relation extraction	AnaMeta [He *et al.*, 2023] GitTables [Hulsebos *et al.*, 2023] TableInstruct [Zhang *et al.*, 2023a] TURL [Deng *et al.*, 2020]	WT	Instruction	Referent entity Column types Relation
Augmentation	Row population Column population Cell filling	TableInstruct [Zhang *et al.*, 2023a] TURL [Deng *et al.*, 2020]	WT	Instruction	Header row Column name Cell
Text-to-SQL	Text-to-SQL	Spider [Yu *et al.*, 2018] BIRD [Li *et al.*, 2023c] Dr.Spider [Chang *et al.*, 2023]	DB	Question	SQL
Advanced data analysis	Feature engineering Visualization Distribution analysis Machine learning	HumanEval [Chen *et al.*, 2021a] MBPP [Austin *et al.*, 2021] InfiAgent-DABench [Hu *et al.*, 2024] DS-1000 [Lai *et al.*, 2023]	SS/DB	Instruction	Code

Tabular Data Preparation denotes the *Interpretation* and *Augmentation* tasks in Table 1. These tasks are typically carried out by data engineers during data preparation phase.

Advanced Table Analysis refers to the *Text-to-SQL* and *Anvanced data analysis* tasks in Table 1. These tasks cater to data engineers looking for an AI copilot to aid in code completion and boost productivity.

Input and Output

In Table 1, we organize the frequently used datasets and benchmarks corresponding to the table tasks. These tasks generally involve two types of inputs: the knowledge input, reflecting the input table's format, and the user input, which includes the users' questions or instructions. Annotating inputs and outputs helps engineers and product managers better understand these table tasks, facilitating more effective system feature design.

2.4 Benchmarks and Evaluation Methods

Traditional benchmarks, such as WikiTQ, WikiSQL, and Spider, are already widely used and studied. We will not elaborate on those here but rather focus on new benchmarks. Table 2 presents recently proposed datasets and benchmarks, along with their sources, sizes, and metrics. We summarize the features of these new benchmarks as follows:

- **Robustness.** For most table tasks (such as Table QA), swapping rows and columns, or replacing column names with synonyms or abbreviations, should not affect the final results. For a large table (that cannot fit into the LLM's context), the placement of the wanted cell, whether at the beginning, end, or middle of the table, should not affect the query result. To evaluate the robustness, Dr. Spider[Chang *et al.*, 2023], RobuT[Zhao *et al.*, 2023b], and Observatory [Cong *et al.*, 2023] are proposed. RobuT and Observatory reveal that the performance of all table methods degrades when perturbations are introduced, yet close-source LLMs (e.g., GPT-3) exhibit greater robustness.
- **Human Involved Labeling.** Some new benchmarks, targeting advanced data analysis tasks, require extensive manual annotation during the data production process. Recently, researchers have started using AI to automate complex data analysis tasks, such as using Python to plot figures or manipulating spreadsheets. DS-1000 [Lai *et al.*, 2023] and InfiAgent-DABench [Hu *et al.*, 2024] are designed to assess advanced data analysis tasks. They gather

Table 2: Evaluation metrics of different benchmarks. We use the abbreviations as follows. EX:Execution Accuracy, VES:Valid Efficiency Score, EM:Exact Match, ABP:Accuracy by Questions, APSQ:Accuracy Proportional by Subquestions, AUSQ:Accuracy Uniform by Subquestions, P:Precision, R:Recall

Dataset	Sources	Size	Metric
BIRD	Kaggle,CTU Prague Open tables	95 DBs 12,751 text-to-SQL pairs	VES,EX
Dr.Spider	Spider	200 DBs 15K perturbed examples	EX,EM
SheetCopilot	superuser.com	28 SSs 13k QA pairs	Pass@k,Exec@k
DS-1000	StackOverflow	451 problems	Pass@1
InfiAgent-DABench	GitHub	631 CSVs 5131 samples	ABQ,APSQ,AUSQ
AnaMeta	public Web T2D,TURL SemTab	467k WT/SSs	F1,R,P
GitTables	GitHub	1M CSVs	F1

data from the internet and annotate it either automatically or semi-automatically. DS-1000 collects questions from StackOverflow, assesses their usefulness manually, and curates a selection of them to form the benchmark. The authors manually adapt the original questions by providing input and output context into test cases, rewriting problems to prevent the data from being learned and memorized by LLMs, and implementing multi-criteria executables. InfiAgent-DABench invites human experts to evaluate the dataset quality and compare human-made and GPT-4 generated data analysis questions via multiple metrics.

- **Larger Scale.** AnaMeta [He *et al.*, 2023] is a large-scale table metadata dataset, and GitTables [Hulsebos *et al.*, 2023] downloads millions of CSV tables from GitHub and aligns them with knowledge bases. These two datasets can evaluate whether the AI system could understand the table schema and benchmark tasks like column type annotation.

Evaluation Metrics

We list the evaluation metric of each benchmark in Table 2. Some benchmarks adopt or improve upon classic evaluation methods. For instance, DS-1000 employs *Pass@K*, a measure frequently used in code generation tasks. Some new benchmarks design their own metrics. For example, InfiAgent-DABench introduced Accuracy by Questions (*ABQ*) and Accuracy Proportional by Subquestions (*APSQ*). If all questions and sub-questions are completed successfully, both of these values would be 1.

Many table tasks have more than one standard answer. Some evaluation pipelines are not yet able to assess the accuracy of different solutions. For example, SheetCopilot, which evaluates spreadsheet operations, can only approximately verify the column names' correctness without inspecting the cells' contents.

3 Taxonomy of LLM for Table

To summarize the existing methods for table tasks, we propose a taxonomy based on two dimensions illustrated in Figure 1. The first dimension is whether the method is based on training or prompting. The second dimension is whether the results are obtained from the model parameters only or relied on agents. With such two-dimensional categorization, Figure 1 comprises four quadrants: *prompting*, *training*, *prompting + agent*, and *training + agent*.

The upper-left quadrant encompasses methods that prompt LLMs directly without the need for external tools. For instance, prompting GPT [Brown *et al.*, 2020] or Codex [Chen *et al.*, 2021a] falls under this category. We delve deeper into training-based approaches in Section 4, prompting techniques for table tasks in Section 5, and LLM-powered agents in Section 6.

4 Tabular LLMs Training

Researchers have proposed various Tabular LMs (TaLMs), wishing these models would handle table characteristics whilst preserving their ability to comprehend text. These methods can be classified into three types: (1) tweaking language models for task-specific fine-tuning, (2) instruction tuning to train a model for various tasks, and (3) retrieval-augmented methods.

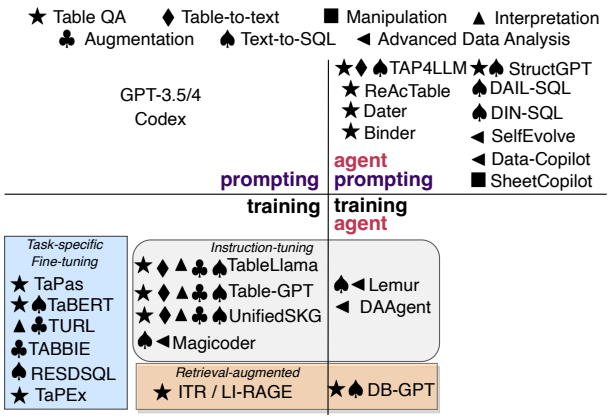

Figure 1: Taxonomy of LLM-based table methods.

4.1 Task-specific Fine-tuning

Task-specific fine-tuning methods modify the model structure, encoding methods, and training objectives to adapt to table tasks. For example, TaPas [Herzig *et al.*, 2020] extends BERT's [Devlin *et al.*, 2019] model architecture and mask language modeling objective to pre-train and fine-tune with tables and related text segments. TaBERT [Yin *et al.*, 2020] encodes a subset of table content most relevant to the input utterance and employs a vertical attention mechanism. TURL [Deng *et al.*, 2020] encodes information of table components (e.g., caption, headers, and cells) into separate input embeddings and fuses them together. TABBIE [Iida *et al.*, 2021] modifies the training objective to detect corrupted cells. TaPEx [Liu *et al.*, 2022] learns a synthetic corpus, which is obtained by automatically synthesizing executable SQL queries and executing these SQLs. RESDSQL [Li *et al.*, 2023a] injects the most relevant schema items into the model during training and ranks schema items during inference to find the optimal one.

Task-tied. These methods are more task-related because they are initially pre-trained/fine-tuned for a specific goal. The foundation models of their methods are relatively small and cannot adapt to various downstream tasks, and these methods require annotated data during fine-tuning.

4.2 Instruction Tuning

Unlike fine-tuning on a single dataset, instruction tuning refers to fine-tuning language models on a collection of datasets. It can substantially improve performance on unseen tasks and offer higher flexibility [Wei *et al.*, 2022a]. Motivated by this feature, there are two types of instruction tuning works. The first type of research focuses on LLMs' understanding of tables, capable of handling various table tasks such as table QA, table-to-text, and interpretation. This type of research utilizes general-purpose foundation LLMs (such as GPT) and a substantial volume of table-related data for instruction tuning. Examples include UnifiedSKG[Xie *et al.*, 2022], TableLlama[Zhang *et al.*, 2023a], and Table-GPT[Li

et al., 2023d]. This approach is referred to as "table instruction tuning". The second type of research, from a code generation perspective, considers tasks requiring generating code, such as text-to-SQL and advanced data analysis, as subdomains of code generation. Therefore, this type of research opts for instruction tuning on LLMs specifically designed for programming, such as CodeLlama [Rozière *et al.*, 2023]. Examples include Magicoder [Wei *et al.*, 2023], Lemur [Xu *et al.*, 2024], and DAAgent [Hu *et al.*, 2024]. We refer to these as "code instruction tuning". As previous studies [Wei *et al.*, 2022a] have shown data quality is essential to instruction tuning; for table tasks, the core is how to construct high-quality datasets.

Table Instruction Tuning. This type of research constructs instruction tuning datasets by leveraging multiple existing table-related datasets and subsequently fine-tunes these datasets on LLMs. Instances of their datasets are in the form of `(instruction, table, completion)` triples. Each triplet comprises the "`instruction`" specifying the task, the "`table`" presenting metadata and content, and the "`completion`" featuring natural language outputs like table QA answers, text from table-to-text transformations, and resultant tables after manipulation, or a mix of text and tables, enabling the model to perform the task. To build high-quality data, Table-GPT employs a synthesis-then-augment approach for dataset construction. Specifically, the dataset is created using 18 synthesis processes ranging from table QA to augmentation. One example is instructing the model to swap rows or columns, and the `completion` is the swapped table, enabling the model to understand the order of rows/columns. TableLlama and UnifiedSKG employ real data from various existing datasets, such as WikiTQ, or Spider, instead of synthesizing data. These papers demonstrate that after table instruction tuning, LLMs could exhibit robust generalization capabilities and tackle unseen table tasks.

Code Instruction Tuing. This type of research involves constructing code datasets and instruction tuning the model. Magicoder first collects open-sourced code snippets and uses them as seeds to prompt an LLM to generate coding problems and solutions. DAAgent is a series of specialized agent models focused on data analysis. Their instruction tuning dataset is crafted by crawling CSVs from GitHub and generating data analysis questions and solutions based on those CSVs. The authors contend that LLM-powered agents depend on the backend model to both master general-purpose capabilities (e.g., reasoning and planning), which can be learned by NL text, and also to guarantee the grounding of programming skills that should be learned from code. Thus, they build a corpus with a code-to-text ratio of 10:1 to train the model. These studies demonstrate that tuning models with code instruction can improve programming skills and enhance data analysis co-piloting.

4.3 Retrieval-augmented Tuning

Like what DPR [Karpukhin *et al.*, 2020] does for open-domain QA, retrieval-augmented table pipelines consist of two neural components: the retriever, which selects a small subset from the total pool of table candidates, and the reader, which produces answers by analyzing each table candidate. Both components are fine-tuned encoders that encode items (e.g., QA questions and table content) into embeddings. Two research examples are ITR [Lin *et al.*, 2023a] and LI-RAGE [Lin *et al.*, 2023b]; they dissect big tables into subtables and jointly train the retriever and reader. DB-GPT [Xue *et al.*, 2023] is an industrial product supporting various functionalities like retrieval-augmented generation, fine-tuning, and agents.

4.4 Summary and Discussion

Training-based methods offer great control, allowing enterprises to conduct private training and deployment without data leakage to third-party model service providers.

Transferability. Before the advent of LLMs, the task-specific fine-tuning methods mainly involves pre-training/fine-tuning on models like BERT or BART. The success of LLMs and instruction tuning exposed the drawbacks of the original task-specific fine-tuning methods. These task-specific models must design specific neural models and depend on annotated data. Thus, these methods show poor transferability.

Cost and Accuracy. Table instruction tuning on LLMs achieves decent results and shows more generalization ablilty, but two issues can't be ignored: cost and accuracy. The expense of pre-training or fine-tuning a large model is substantial. Even fine-tuning a 7B model requires eight 80GB GPUs, a cost not all institutions can afford. Moreover, the accuracy of instruction tuning methods is not always optimal compared with prompting close-sourced LLMs. For example, on some tasks, the 7B TableLlama can't outperform task-specific fine-tuning models. On data analysis tasks, instruction tuned models like Lemur and DAAgent cannot surpass GPT-4. Table-GPT continued to train on GPT-3.5 and achieved better results on all table-related tasks than GPT-3.5 and ChatGPT. However, the cost of training is prohibitively high for ordinary enterprise users who wish to deploy privately. Regarding training data, the cost of manual annotation is also high. Although synthesis is a cheap choice, the data quality is another concern. A more prevalent approach is using GPT as a teacher model for data annotation. Evidently, GPT is the performance ceiling.

5 Prompting LLMs for Table Tasks

In this sub-section, we mainly discuss the techniques of prompting LLMs for table tasks, specifically table serialization and example selection for few-shot learning. We will leave agent-based approaches in the next sub-section.

5.1 Table Serialization

The first issue is converting tables into prompts while maintaining their semantic integrity. LLMs require prompts in a linear, sequential text format, contrasting with the 2-dimensional structure of tables with a defined schema. The prompt may contain the table content and the table schema.

Table Content. A straightforward yet common way is to linearize tables row by row, inserting column separators

(e.g., "|") between cells. Researchers benchmark evaluated LLMs' performance with various table formats, including CSV, markdown, JSON, XML, and HTML [Sui *et al.*, 2024; Singha *et al.*, 2023]. The results suggest that HTML and NL with separators (markdown or CSV) are the two most effective options, which can be assumed that the training corpora include substantial code and web tables.

Table Schema. [Gao *et al.*, 2023; Nan *et al.*, 2023] explored prompt representation methods for text-to-SQL tasks. In text-to-SQL tasks, the prompt representation should include the NL question, table schemas, instructions, etc. Table schemas can be represented in plain text or coded forms using `CREATE TABLE` statements. Foreign keys can suggest the relationships among different relational tables. Special instructions may stipulate that LLMs should not provide verbose responses with explanations. In this way, the shorter final responses from LLMs align more closely with the standard answers in the benchmarks. Results [Gao *et al.*, 2023; Nan *et al.*, 2023] show that information about foreign keys and the "*with no explanation*" instruction rule can benefit the text-to-SQL task.

5.2 Few-shot Example Selection

In-context learning involves selecting and organizing the most helpful demonstrative examples into the prompt. For the text-to-SQL task, selecting the most relevant example can be achieved by choosing examples more related to the target NL question or examples more similar to the potential SQL. Nan et al. [Nan *et al.*, 2023] argue that diversity should be considered in addition to similarity. As the context length is limited, organizing all the information (i.e., NL question, schema, SQL) into the prompt ensures the quality of examples but sacrifices quantity. DAIL-SQL [Gao *et al.*, 2023] proposes a method that balances the quality and quantity by removing examples' schemas, which are token-cost.

6 LLM-powered Table Agents

LLMs struggle with functionalities, such as arithmetic or factual lookups, which are essential for table tasks. They cannot interact with the environment, like manipulating data in tables. LLM-powered agents can help solve complex table tasks. Agents decompose complex tasks into subtasks, plan actions that use external tools, and can reflect upon or refine previous actions. Research like ReAct [Yao *et al.*, 2023] constructs an iterative closed loop for agent-based systems. For table tasks, as depicted in Figure 2, the agent first observes the table data and user intent, generates prompts, decomposes complex tasks, plans actions, executes them on the table environment, and then updates the observation. The process is repeated until expectations are met.

6.1 Complex Task Decomposition

Inspired by the CoT and least-to-most prompting methods, researchers instruct LLMs to decompose complex table tasks into simpler sub-tasks. DIN-SQL [Pourreza and Rafiei, 2023] proposes breaking down the text-to-SQL task into subtasks. The process involves identifying the relevant tables

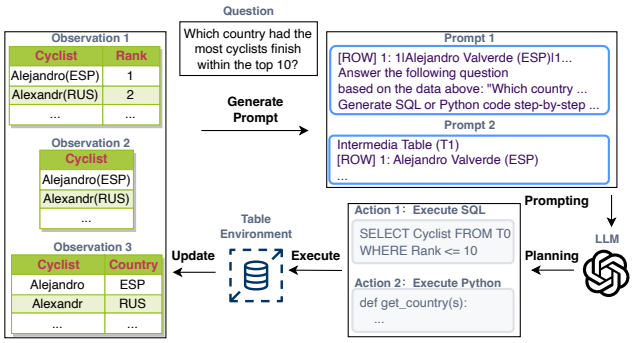

Figure 2: The iterative process of an LLM-powered agent system for table tasks.

and columns associated with the query and producing intermediate sub-queries. For web tables, Dater [Ye *et al.*, 2023] exploit LLMs as decomposers by breaking down huge evidence (a huge table) into sub-evidence (a small table) and decomposing a complex question into simpler sub-questions and getting SQLs.

6.2 Action Definition

In table tasks, actions function as the intermediary between LLMs and software tools like Excel or Python interpreters. Rather than simply translating software APIs into natural language (NL) text, actions should abstract and categorize similar API types for better clarity and utility. Moreover, for complex APIs with numerous arguments, actions should be accompanied by a usage guide to minimize errors in action generation by LLMs. Given the varied and intricate nature of software APIs, we categorize action definition strategies into two approaches: one employs clustering methods to define actions, while the other relies on manual definition.

Clustering. For complex tasks like table manipulation and advanced data analysis, the variety of actions increases, and each action requires more arguments and parameters. Thus, it's crucial to carefully abstract a set of actions. Sheet-Copilot [Li *et al.*, 2023b], an agent system for spreadsheets, models existing spreadsheet software APIs into atomic actions. An atomic action comprises the API name, argument lists, document, etc. The authors design these atomic actions by crawling spreadsheet-related questions online, embedding and clustering them into categories, and abstracting them to software APIs by choosing the most representative ones. Data-Copilot [Zhang *et al.*, 2023b] is for data science and visualization. Its actions are two-level: the interface is high-level pseudo-code descriptions, and the code is low-level executable. It designs actions by autonomously exploring the data to get seed user requests, prompting an LLM with seed requests to get more versatile interfaces, merging similar interfaces, and generating code.

Manually Defined. For tasks like table QA and text-to-SQL, the diversity of actions is not so high and can be manually defined. In Binder [Cheng *et al.*, 2023], actions are two types of modifiable program templates: Python and SQL. These templates can be updated based on previous observations and LLM responses. ReAcTable [Zhang *et al.*, 2023c] incorporates three kinds of actions: (1) generating a SQL query, (2)

generating Python code, and (3) directly answering the question. It extends the ReAct framework with an *observation-action-reflection* loop specifically for table tasks. Within this iterative loop, it progressively refines the table data by generating and executing an SQL query to retrieve table knowledge. If the existing table lacks the required information or if the answer cannot be directly queried via an SQL query, it generates and executes Python code to produce an intermediate table, thereby filling missing information into an intermediate table.

6.3 Reflection and Revision

LLMs often generate answers "without thinking", while agents can try different approaches, vote to select the best one, and even reflect on past actions, learn from mistakes, and refine them for future steps.

Self-consistency and Voting. Generating multiple reasoning paths and choosing the most consistent answer can significantly enhance LLM's accuracy on complex tasks [Wang *et al.*, 2023b]. This has also been demonstrated in table tasks. For instance, Dater, DAIL-SQL, and ReAcTable report that self-consistency can improve the accuracy of LLMs on table QA and text-to-SQL tasks. However, both DAIL-SQL and ReAcTable argue that while self-consistency and voting enhance accuracy, these methods are extremely time-consuming, and the cost is notably higher, especially considering that prompting LLMs is expensive.

Revising. SheetCopilot aims for spreadsheet manipulation, where a mistake in one step's action can significantly impact the following analysis and processing. To handle this challenge, SheetCopilot adopts a mechanism that reflects and refines past actions. Specifically, it employs a proposing stage that validates the proposed actions and a revising stage to refine them. These techniques prevent error actions like wrong arguments. SelfEvolve [Jiang *et al.*, 2023b] achieves decent results on the Python data analysis task by asking the LLM to perform debugging if the generated code cannot execute in the Python interpreter.

6.4 Framework for Multiple Tasks

While most agents are for a specific task, there are frameworks for multiple tasks. StructGPT [Jiang *et al.*, 2023a] is capable of solving both table QA and text-to-SQL problems by developing three types of actions that target web tables, databases, and knowledge graphs. These actions function as a reader, extracting knowledge from the data, which is then used to assist the LLM in its future planning. The actions for web tables include extracting data or column names, whereas for databases, the actions involve extracting table data or metadata. In addition to solving the aforementioned problems, TAP4LLM [Sui *et al.*, 2023] also aims to perform Table Augmentation. It primarily selects appropriate rows and columns from the table (called *Table Sampling*), integrates them with other table data (relevant external knowledge, metadata) (called *Table Augmentation*), and serializes this information to fit the context length of the LLM.

6.5 Summary and Disscussion

The LLM-powered agent method combines the power of LLMs and the flexibility of external tools.

Limited Transferability. However, these approaches often necessitate hard-coded prompt templates, which are long strings of instructions and demonstrations manually crafted through trial and error. A given string prompt might not generalize well to other pipelines, LLMs, domains, or data inputs. Thus, it has limited transferability.

Cost. Agents must prompt LLMs repeatedly to fulfill users' expectations. Consequently, this increases the time and financial costs.

Privacy Issue. Most prompting methods involve requesting third-party model service providers, such as OpenAI, which could lead to data leakage, a situation many enterprises are unacceptable.

7 Conclusion and Discussion

This paper reviews table processing tasks, benchmarks, and LLM-based methods. Although LLM-based methods effectively solve some table tasks, several challenges remain unaddressed.

Instruction-tuning with Agent could be Potential Solution for Private Deployment. Instruction-tuning a foundation model based on open-source LLMs offers certain advantages. Works such as Table-GPT and Lemur have demonstrated the flexibility of instruction-tuning, achieving commendable results on various downstream tasks. These models can be deployed privately, preventing the leakage of sensitive corporate data to third-party model service providers, and can be further powered by agents. However, research about agents on open-source LLMs is still limited. Several studies have already demonstrated that, for table tasks, prompting closed-source LLMs surpasses open-source LLMs in terms of accuracy and robustness. Agents heavily rely on the underlying LLM to understand user intent and plan actions, so the effectiveness of these methods when transferred to closed-source LLMs remains to be further studied.

Prompting and Agent-based Method Could be Simplified. Existing prompting-based methods are too complicated and time-consuming. As LLMs are highly sensitive to prompts, existing prompts are implemented using hard-coded "prompt templates". These prompting-based methods are not scalable and cannot easily adopted into production environments. Moreover, current methods require repeated interaction with LLMs or frequent use of tools, which is very time-consuming.

Benchmarks Need to Incorporate Real-world Requirements. Firstly, tables in real-world scenarios often lack well-defined semantics, making it challenging for LLMs to comprehend their schema and content accurately. Secondly, the absence of a unified schema definition across real-world tables complicates the task of extracting information from heterogeneous table sources. Lastly, processing real-world tables often demands domain-specific knowledge, like deriving formulas from financial insights.

References

[Austin *et al.*, 2021] Jacob Austin, Augustus Odena, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[Badaro *et al.*, 2023] Gilbert Badaro, Mohammed Saeed, et al. Transformers for tabular data representation: A survey of models and applications. *TACL*, 2023.

[Brown *et al.*, 2020] Tom Brown, Benjamin Mann, et al. Language Models are Few-Shot Learners. In *NeurIPS*, 2020.

[Chang *et al.*, 2023] Shuaichen Chang, Jun Wang, Mingwen Dong, et al. Dr. spider: A diagnostic evaluation benchmark towards text-to-sql robustness. In *ICLR*, 2023.

[Chen *et al.*, 2020a] Wenhu Chen, Hongmin Wang, et al. TabFact: A large-scale dataset for table-based fact verification. In *ICLR*, 2020.

[Chen *et al.*, 2020b] Wenhu Chen, Hanwen Zha, et al. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *EMNLP*, 2020.

[Chen *et al.*, 2021a] Mark Chen, Jerry Tworek, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[Chen *et al.*, 2021b] Xinyun Chen, Petros Maniatis, et al. Spreadsheetcoder: Formula prediction from semi-structured context. In *ICML*, 2021.

[Cheng *et al.*, 2023] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, et al. Binding language models in symbolic languages. In *ICLR*, 2023.

[Cong *et al.*, 2023] Tianji Cong, Madelon Hulsebos, Zhenjie Sun, et al. Observatory: Characterizing Embeddings of Relational Tables. In *VLDB*, 2023.

[Deng *et al.*, 2020] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. TURL: Table understanding through representation learning. In *VLDB*, 2020.

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 2019.

[Dong *et al.*, 2022] Haoyu Dong, Zhoujun Cheng, et al. Table Pre-training: A Survey on Model Architectures, Pre-training Objectives, and Downstream Tasks. In *IJCAI*, 2022.

[Gao *et al.*, 2023] Dawei Gao, Haibin Wang, et al. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *arXiv preprint arXiv:2308.15363*, 2023.

[He *et al.*, 2023] Xinyi He, Mengyu Zhou, et al. Anameta: A table understanding dataset of field metadata knowledge shared by multi-dimensional data analysis tasks. In *ACL*, 2023.

[Herzig *et al.*, 2020] Jonathan Herzig, Pawel Krzysztof Nowak, et al. TaPas: Weakly Supervised Table Parsing via Pre-training. In *ACL*, 2020.

[Hu *et al.*, 2024] Xueyu Hu, Ziyu Zhao, Shuang Wei, et al. InfiAgent-DABench: Evaluating Agents on Data Analysis Tasks. *arXiv:2401.05507*, 2024.

[Hulsebos *et al.*, 2023] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. Gittables: A large-scale corpus of relational tables. In *SIGMOD*, 2023.

[Iida *et al.*, 2021] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. Tabbie: Pretrained representations of tabular data. *arXiv preprint arXiv:2105.02584*, 2021.

[Iyyer *et al.*, 2017] Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. Search-based neural structured learning for sequential question answering. In *ACL*, 2017.

[Jiang *et al.*, 2023a] Jinhao Jiang, Kun Zhou, et al. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In *EMNLP*, 2023.

[Jiang *et al.*, 2023b] Shuyang Jiang, Yuhao Wang, and Yu Wang. SelfEvolve: A Code Evolution Framework via Large Language Models. *arXiv preprint arXiv:2306.02907*, 2023.

[Jin *et al.*, 2022] Nengzheng Jin, Joanna Siebert, Dongfang Li, and Qingcai Chen. A survey on table question answering: Recent advances. In *CCKS*, 2022.

[Karpukhin *et al.*, 2020] Vladimir Karpukhin, Barlas Oguz, Sewon Min, et al. Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP*, 2020.

[Lai *et al.*, 2023] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, et al. Ds-1000: A natural and reliable benchmark for data science code generation. In *ICML*, 2023.

[Li *et al.*, 2023a] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. RESDSQL: Decoupling schema linking and skeleton parsing for text-to-SQL. In *AAAI*, 2023.

[Li *et al.*, 2023b] Hongxin Li, Jingran Su, et al. SheetCopilot: Bringing Software Productivity to the Next Level through Large Language Models. In *NeurIPS*, 2023.

[Li *et al.*, 2023c] Jinyang Li, Binyuan Hui, Ge Qu, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *NeurIPS*, 2023.

[Li *et al.*, 2023d] Peng Li, Yeye He, Dror Yashar, et al. Table-GPT: Table-tuned GPT for Diverse Table Tasks. *arXiv preprint arXiv:2310.09263*, 2023.

[Lin *et al.*, 2023a] Weizhe Lin, Rexhina Blloshmi, Bill Byrne, et al. An inner table retriever for robust table question answering. In *ACL*, 2023.

[Lin *et al.*, 2023b] Weizhe Lin, Rexhina Blloshmi, Bill Byrne, et al. Li-rage: Late interaction retrieval augmented generation with explicit signals for open-domain table question answering. In *ACL*, 2023.

[Liu *et al.*, 2022] Qian Liu, Bei Chen, et al. TAPEX: Table pre-training via learning a neural SQL executor. In *ICLR*, 2022.

[Nan *et al.*, 2021] Linyong Nan, Dragomir Radev, et al. DART: Open-domain structured data record to text generation. In *NAACL*, 2021.

[Nan *et al.*, 2022] Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, et al. Fetaqa: Free-form table question answering. *TACL*, 2022.

[Nan *et al.*, 2023] Linyong Nan, Yilun Zhao, et al. Enhancing Text-to-SQL Capabilities of Large Language Models: A Study on Prompt Design Strategies. In *Findings of EMNLP*, 2023.

[Parikh *et al.*, 2020] Ankur Parikh, Xuezhi Wang, et al. ToTTo: A controlled table-to-text generation dataset. In *EMNLP*, 2020.

[Pasupat and Liang, 2015] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *ACL*, 2015.

[Pourreza and Rafiei, 2023] Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *NeurIPS*, 2023.

[Qin *et al.*, 2022] Bowen Qin, Binyuan Hui, Lihan Wang, et al. A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions. *arXiv preprint arXiv:2208.13629*, 2022.

[Raffel *et al.*, 2020] Colin Raffel, Noam Shazeer, Adam Roberts, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 2020.

[Rahman *et al.*, 2020] Sajjadur Rahman, Kelly Mack, et al. Benchmarking Spreadsheet Systems. In *SIGMOD*, 2020.

[Rozière *et al.*, 2023] Baptiste Rozière, Jonas Gehring, et al. Code Llama: Open Foundation Models for Code. *arXiv preprint arXiv:2308.12950*, 2023.

[Singha *et al.*, 2023] Ananya Singha, José Cambronero, et al. Tabular representation, noisy operators, and impacts on table structure understanding tasks in LLMs. In *NeurIPS TRL Workshop*, 2023.

[Sui *et al.*, 2023] Yuan Sui, Jiaru Zou, et al. TAP4LLM: Table Provider on Sampling, Augmenting, and Packing Semi-structured Data for Large Language Model Reasoning. *arXiv preprint arXiv:2312.09039*, 2023.

[Sui *et al.*, 2024] Yuan Sui, Mengyu Zhou, et al. GPT4Table: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study. In *WSDM*, 2024.

[Wang *et al.*, 2023a] Lei Wang, Chen Ma, et al. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023.

[Wang *et al.*, 2023b] Xuezhi Wang, Jason Wei, et al. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023.

[Wei *et al.*, 2022a] Jason Wei, Maarten Bosma, et al. Fine-tuned language models are zero-shot learners. In *ICLR*, 2022.

[Wei *et al.*, 2022b] Jason Wei, Xuezhi Wang, et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

[Wei *et al.*, 2023] Yuxiang Wei, Zhe Wang, et al. Magicoder: Source Code Is All You Need. *arXiv preprint arXiv:2312.02120*, 2023.

[Xie *et al.*, 2022] Tianbao Xie, Chen Henry Wu, et al. UnifiedSKG: Unifying and Multi-Tasking Structured Knowledge Grounding with Text-to-Text Language Models. In *EMNLP*, 2022.

[Xu *et al.*, 2024] Yiheng Xu, Hongjin Su, et al. Lemur: Harmonizing Natural Language and Code for Language Agents. In *ICLR*, 2024.

[Xue *et al.*, 2023] Siqiao Xue, Caigao Jiang, Wenhui Shi, et al. Db-gpt: Empowering database interactions with private large language models. *arXiv preprint arXiv:2312.17449*, 2023.

[Yao *et al.*, 2023] Shunyu Yao, Jeffrey Zhao, et al. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

[Ye *et al.*, 2023] Yunhu Ye, Binyuan Hui, et al. Large Language Models are Versatile Decomposers: Decomposing Evidence and Questions for Table-based Reasoning. In *SIGIR*, 2023.

[Yin *et al.*, 2020] Pengcheng Yin, Graham Neubig, et al. Tabert: Pretraining for joint understanding of textual and tabular data. In *ACL*, 2020.

[Yu *et al.*, 2018] Tao Yu, Rui Zhang, et al. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP*, 2018.

[Zhang and Balog, 2020] Shuo Zhang and Krisztian Balog. Web Table Extraction, Retrieval, and Augmentation: A Survey. *ACM TIST*, 2020.

[Zhang *et al.*, 2023a] Tianshu Zhang, Xiang Yue, and Huan Sun. TableLlama: Towards Open Large Generalist Models for Tables. *arXiv preprint arXiv:2311.09206*, 2023.

[Zhang *et al.*, 2023b] Wenqi Zhang, Yongliang Shen, et al. Data-Copilot: Bridging Billions of Data and Humans with Autonomous Workflow. *arXiv preprint arXiv:2306.07209*, 2023.

[Zhang *et al.*, 2023c] Yunjia Zhang, Jordan Henkel, et al. ReAcTable: Enhancing ReAct for Table Question Answering. *arXiv preprint arXiv:2310.00815*, 2023.

[Zhao *et al.*, 2023a] Wayne Xin Zhao, Kun Zhou, et al. A Survey of Large Language Models. *arXiv preprint arXiv:2303.18223*, 2023.

[Zhao *et al.*, 2023b] Yilun Zhao, Chen Zhao, et al. RobuT: A Systematic Study of Table QA Robustness Against Human-Annotated Adversarial Perturbations. In *ACL*, 2023.

[Zhong *et al.*, 2017] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.