Front. Comput. Sci., 2025, 19(2): 192350
https://doi.org/10.1007/s11704-024-40763-6

REVIEW ARTICLE

Large language model for table processing: a survey

Weizheng LU[1], Jing ZHANG (✉)[1], Ju FAN[1,2], Zihao FU[3], Yueguo CHEN (✉)[1,2], Xiaoyong DU[1,2]

1 School of Information, Renmin University of China, Beijing 100872, China
2 Key Laboratory of Data Engineering and Knowledge Engineering, Beijing 100872, China
3 WPS Office, Kingsoft Co., Zhuhai 519080, China

Abstract Tables, typically two-dimensional and structured to store large amounts of data, are essential in daily activities like database queries, spreadsheet manipulations, Web table question answering, and image table information extraction. Automating these table-centric tasks with Large Language Models (LLMs) or Visual Language Models (VLMs) offers significant public benefits, garnering interest from academia and industry. This survey provides a comprehensive overview of table-related tasks, examining both user scenarios and technical aspects. It covers traditional tasks like table question answering as well as emerging fields such as spreadsheet manipulation and table data analysis. We summarize the training techniques for LLMs and VLMs tailored for table processing. Additionally, we discuss prompt engineering, particularly the use of LLM-powered agents, for various table-related tasks. Finally, we highlight several challenges, including diverse user input when serving and slow thinking using chain-of-thought.

Keywords data mining and knowledge discovery, table processing, large language model

1 Introduction

In this data-driven era, a substantial volume of data is structured and stored in the form of tables [1,2]. Everyday tasks involving tables, such as database queries, spreadsheet manipulations, question answering on Web tables, and information extraction from image 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 table-related tasks provides substantial benefits to the general public, garnering significant interest from both academic and industrial sectors [2−4].

Recently, large language models (LLMs) have demonstrated their effectiveness and versatility across diverse tasks, leading to significant advancements in natural language processing [5]. This success has spurred researchers to investigate the application of LLMs to table-related tasks. However, the structure of tables differs from the plain text [6] typically used during LLM pre-training.

- **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.

Although many text-related tasks, such as those in STEM (Science, Technology, Engineering, and Mathematics) fields, require complex reasoning and external tools, table processing tasks are different due to the structural nature of tables and the user intent of querying knowledge from tables. For instance, LLMs need to understand table schemas, locate data within two-dimensional tables, and execute SQL queries to retrieve data. The unique challenges presented by table processing tasks emphasize the need to tailor LLMs for these specific purposes. Early research, such as TaBERT [7], TaPas [8], TURL [9], and TaPEx [10], 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

Received July 26, 2024; accepted November 3, 2024

E-mail: zhang-jing@ruc.edu.cn; chenyueguo@ruc.edu.cn

sufficiently large or versatile to handle a broader range of table tasks. Latest LLM-based approaches tackle table tasks in two primary ways: (1) curating table datasets and pre-train or fine-tune a table model [11−13]; (2) prompting an LLM or building an LLM-powered agent by utilizing the LLM's strong reasoning ability to understand table data [14−17]. These newer methods leverage LLM-specific technologies, such as instruction-tuning [18], in-context learning [19], chain-of-thought reasoning [20], and autonomous agents [21], showcasing a more versatile and comprehensive approach to table processing.

Taxonomy The goal of this survey is to offer a comprehensive review of technological advancements in LLM for table processing and to summarize current research directions. As depicted in Fig. 1, we have categorized the literature into a taxonomy of four key categories: table types and table tasks, table data representation, table training, and table prompting. These four categories cover distinct and interrelated research topics, offering a systematic and comprehensive review of LLM for table processing research.

Contribution The main contribution of this survey is its extensive coverage of a wide range of table tasks, including recently proposed spreadsheet manipulation and data analysis. We discuss table tasks not only from a technical perspective but also from the table data lifecycle and from the end-user's viewpoint. We categorize methods based on the latest paradigms in LLM usage, focusing on instruction-tuning, data synthesis, chain-of-thought, ReAct, and LLM-powered agent approaches. We compile recent datasets, benchmarks, and training corpora. We collect resources such as papers, code, and datasets, which can be accessed at our website [22].

Comparison with related surveys Earlier surveys, such as those by Dong et al. [1] and Badaro et al. [2], primarily concentrate on pre-training or fine-tuning techniques using smaller models like BERT [7,8] or BART [10]. However, they do not address methods based on LLMs, particularly those involving prompting strategies and agent-based approaches. Additionally, some surveys are confined to limited table tasks. For instance, Jin et al. [23] focus solely on table QA. Zhang et al. [4] focus on table reasoning, overlooking tasks such as spreadsheet manipulation. Fang et al. [3] review research on table data prediction, generation, and understanding; however, the discussion on table processing lacks depth. Qin et al. [24] and Hong et al. [25] concentrate on natural language to SQL (NL2SQL), overlooking spreadsheet manipulation and data analysis tasks.

2 Table types and table tasks

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

2.1 Table definition

This paper mainly focuses on (semi-)structured tables, which are grids of cells arranged in rows and columns [26]. Every column in a table signifies a specific attribute, each having its own data type (e.g., numeric, string, or date). 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. [1]. The two-dimensional structure endows the data with a schema, which includes column names, data types, constraints, and relationships with other tables. Tables are mainly stored and presented in the following formats:

- **Spreadsheet** (SS): Spreadsheets are used by one-tenth of the global population [27], with Google Sheets and Microsoft Excel being the two most popular systems. Spreadsheet systems allow users to format tables by customizing styles (such as font or color) to display or highlight certain data. Spreadsheet tables often feature irregular layouts with merged cells, hierarchical columns, and annotations, as shown in Fig. 2(a). The irregular layout is designed to improve human understanding of the table data, but it makes machine parsing difficult. Spreadsheet systems provide features like sorting, filtering, formulas, data visualization, or table programming (e.g., Visual Basic for Applications, VBA) for automated or semi-automated data analysis. Typical applications of spreadsheets include: teachers in schools recording grades, human-resource departments recording employee information, and sales personnel tracking sales data, among others.

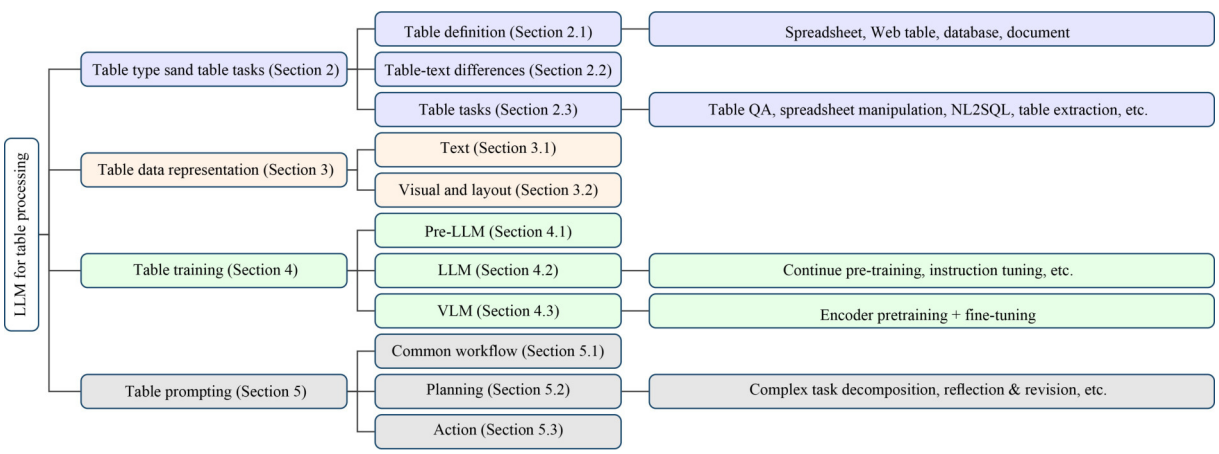


Fig. 1 Taxonomy of LLMs for table processing

Fig. 2 Four types of tables: spreadsheet, Web table, database, and document. (a) Spreadsheet; (b) Web table; (c) database; (d) document

- **Web table** (WT): The Web hosts a multitude of tables, created for diverse purposes and containing valuable information [26,28,29]. These tables exist in various formats, from HTML to markdown, JSON, and XML. As Web tables are embedded in Web pages, they have much contextual information, such as the page's title, the surrounding text, and so on. Web tables can also have various styles and embed URLs. A well-known Web table example is the table on a Wikipedia page shown in Fig. 2(b). Web tables contain substantial factual knowledge, and the community has been working to extract and structure these tables. For example, the WDC (Web Data Commons) Web Table Corpus project converts billions of Web pages from the Common Crawl corpus into structured tables [26], and Bhagavatula et al. [29] extract millions of tables from Wikipedia. The extraction process involves tasks like entity linking or relation extraction, and the extracted tables can be utilized for question answering.
- **Database** (DB): Tables in relational databases are highly structured, i.e., each database table is explicitly defined with a schema at the creation time. Users should use SQL to interact with database tables. Database systems are categorized into online transaction processing (OLTP) and online analytic processing (OLAP). OLTP systems frequently utilize foreign keys to establish relationships between tables, as illustrated in Fig. 2(c). Tables in OLAP systems or data warehouses, often comprising many columns, are called wide tables. Wide tables can minimize the need for complex joins. Database systems are scalable and robust and are employed in enterprises requiring data accuracy and integrity. Companies typically employ an IT team proficient in programming to manage these database systems.
- **Document** (DOC) : There is another category of tables embedded in various document formats, such as images (*.png*, *.jpg*, etc.), PDFs (*.pdf*), or Microsoft Word files (*.docx*). Common examples of documents with tables include purchase orders, financial reports, sales contracts, receipts, academic papers (Fig. 2(d)), and numerous other types. Users want to extract tables from these documents, structure them, and convert them into table-native formats (spreadsheet or HTML). These tables are often surrounded by text, necessitating identifying their location before extracting their content, and the layout of these tables may be irregular. Furthermore, unlike ordinary images or plain text documents, document-embedded tables rely on an exact two-dimensional coordinate system. Any misalignment in the rows and columns can significantly impair the understanding of the information presented.

Since these four types of tables are tailored to different user scenarios and address diverse problems, integrating artificial intelligence (AI) models with these four types of tables varies accordingly. Spreadsheet systems aim to copilot users by automating various manipulation operations. Web tables can be used for table QA. Databases now widely incorporate NL2SQL technologies, aiding human engineers in data engineering and analytical tasks. Tables within documents need to be identified, structured, and transformed into table-native formats.

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. [12] 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) some of them remain unaffected by row and column permutations.

2.3 Table tasks

Table 1 summarizes table tasks that can be automated by LLMs. We also list the description of the table task, along with one or two related works, the types of tables addressed, and the datasets used.

Table QA and fact verification are the most traditional table tasks, which extract knowledge from tables to answer natural language (NL) questions. Table-to-text produces an NL text based on table data. Data cleaning identifies and corrects errors in table data. Column/Row/Cell population generates possible column/row/cell for a table. Entity linking disambiguates specific entities mentioned, while column type annotation categorizes columns with types from knowledge bases. These two tasks often utilize external knowledge bases. Spreadsheet systems are originally designed for human users. Spreadsheet manipulation is a task that leverages AI to modify spreadsheets automatically, where AI accesses spreadsheet systems' APIs or formulas. NL2SQL translates NL questions into SQL queries and can improve the efficiency of data analysts when writing SQL queries. This task has been extensively studied for years, and LLMs enhance accuracy in this field. Data analysis consists of feature engineering, machine learning, etc. Table detection identifies tables within

Table 1 Table tasks, input types, descriptions (related work), and representative datasets. In the Table Type column, the abbreviations are as follows: WT for Web table, SS for spreadsheet, DB for database, and DOC for document

Task name	Table type	Description (related work)	Example dataset
Table QA	WT	Answer a NL question given a table ([30,31])	WikiTableQuestion [32]
Table fact verification	WT	Verifying facts given a table ([31,33])	TabFact [34]
Table-to-text	WT	Produce a NL question given a table ([11])	ToTTo [35]
Data cleaning	WT/SS/DB	Correct errors of table data ([36,37])	–
Column/Row/Cell population	WT/SS/DB	Populate possible column/row/cell for a table ([11,12])	TURL [9]
Entity linking	WT	Link the selected entity to the knowledge base ([11,12])	TURL [9]
Column type annotation	WT	Choose types for the column in the table ([11,12])	TURL [9]
Spreadsheet manipulation	SS	Manipulate spreadsheets ([16,38])	SpreadsheetBench [39]
NL2SQL	DB	Translate a NL question to a SQL query ([40,41])	Spider [42]
Data analysis	SS/DB	Table data analysis pipeline, consists of feature engineering, machine learning, etc. ([43,44])	DS-1000 [45]
Table detection	DOC	Locate tables in documents ([46])	TableBank [47]
Table extraction	DOC	Extract and structuralize tables from documents ([46,48])	PubTabNet [49]

documents, while table extraction converts them into table-native formats such as markdown, HTML, or spreadsheet. The tasks mentioned above can broadly be categorized into table-related, spreadsheet-related, database-related, and document-related tasks. These tasks require AI models to directly understand table contents, write code to manipulate spreadsheets, write SQL to access databases, or extract table data from documents.

2.4 Data lifecycle and end-users' perspective

Researchers often concentrate 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 benchmark results. To meet end-users' needs, the industry focuses on developing products and tools for them.

2.4.1 Data lifecycle

End-users' requirements vary based on their roles; common users typically need table querying and manipulation capabilities, while data engineers need data preparation and modeling tools. Different end users are at different stages of the data lifecycle. We divide the table data lifecycle into the following five stages: Data Entry, Data Cleaning, Data CRUD (Create, Read, Update, and Delete), Data Analysis, and Data Visualization. Figure 3 shows the five stages in table data processing, with corresponding table tasks annotated below.

Data entry consists of two parts, one is helping users to create the table structure, and the other is the precise entry of data by converting unstructured data formats into (semi-) structured tables. When creating a table, LLMs can help list the possible column headers. For instance, Google Sheets offers a feature that generates a new table with suggested column headers and example table data. Another application scenario involves converting tables from images or PDFs into formats natively suited for tables, facilitating subsequent stages of table processing. This feature requires AI systems capable of multimodal table understanding [13,46], and systems like ChatGPT-4o can now convert table images into structured formats.

Data cleaning identifies and corrects errors, inaccuracies, missing values, and duplicates in a table dataset to improve its quality and reliability for further analysis [50]. This stage needs to identify erroneous parts and impute errors or missing values, utilizing techniques like cell population or column type annotation.

Data CRUD includes the following tasks: table QA, table fact verification, NL2SQL, and spreadsheet manipulation. This stage involves querying Web table knowledge, transforming upstream database tables into downstream tables in data warehouses or data lakes, or managing spreadsheet tables by calling the system's APIs or formulas. In this stage, AI systems usually enable individuals to process tables through NL questions or instructions.

Data analysis includes feature engineering, outlier detection, machine learning, visualization, etc. Thus, it requires higher intelligence, as it involves understanding table data, having some domain knowledge, and utilizing tools (e.g., SQL, Python, or VBA) to model tables and give insights.

Data visualization is an essential step to improve the expressiveness of data. Different data types paired with distinctive chart types will show completely different expressiveness. Users expect that AI systems can automatically select the best chart type and graph descriptions.

2.4.2 End-users' experience

As shown in Fig. 4, from the perspective of product design and usage, the interaction between end-users and AI systems can be categorized into three types: Beside, Inside, and Outside. "Beside" refers to adding a copilot beside the application. "Inside" indicates that AI is the core component

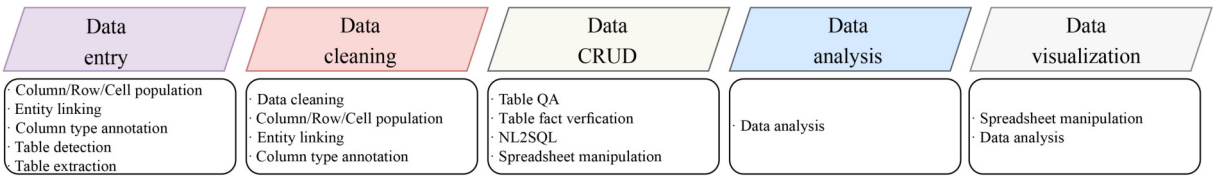


Fig. 3 Table processing lifecycle with table tasks annotated

(a) (b) (c)

Fig. 4 Three types of human interaction with table AI system. (a) Beside; (b) inside; (c) outside

of the product. "Outside" implies that the AI system orchestrates across different applications and tasks. No matter the type, users tend to interact directly with AI through NL, and the quality of the user experience is strongly linked to the underlying table AI system [51].

3 Table data representation

When processing tables and table tasks, AI systems must first convert tables into machine-readable representations. Modern neural networks, which are widely adopted in AI systems, need to encode text or images into numerical representations or embeddings, and then perform computations on these embeddings. Consequently, we must format table data accordingly and feed it into language or visual language models. This section focuses on the serialization of text tables and the processing of document tables.

3.1 Text representation

LLMs require prompts in a linear, sequential text format, contrasting with the two-dimensional structure of tables with a defined schema. So, we should maintain their semantic integrity when converting tables into prompts. The prompt may contain the table content and the table schema.

Table content A straightforward yet common way is to linearize tables into markdown format, i.e., separating rows by new lines and inserting column separators (e.g., "|") between cells. Several studies [14,52] evaluate different table serialization formats to assess whether LLMs accurately understand structured tables. They compare formats like CSV, markdown, JSON, HTML and pandas DataFrame. The evaluation 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. Spreadsheets may contain merged cells and hierarchical columns, meaning simple row-by-row and column-by-column serialization is insufficient. Tian et al. [53] argue that homogeneous rows or columns in spreadsheets contribute little to understanding their layout and structure. Consequently, they introduce an anchor-based approach that pinpoints heterogeneous rows and columns as anchors. Subsequently, they utilize an inverted-index style encoding technique to convert cell locations, values, and schemas into a JSON dictionary format.

Table schema Some research [41,54] explores schema representation methods for NL2SQL tasks. In NL2SQL tasks, the prompt 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 prompt rules like "*with no explanation*" force LLMs to provide clear and concise responses that align more closely with the standard answers in the benchmarks. Results [41,54] show that information about foreign keys and the rules like "*with no explanation*" instruction can benefit the NL2SQL task.

Text embedding Like other text data, once text-based tables are serialized, the serialized table data will be embedded by LLMs.

3.2 Visual and layout representation

Web tables, spreadsheets, and document-embedded tables may have visual cues, such as color highlighting, which could potentially aid VLMs in identifying accurate table information. Deng et al. [55] explore the capability of VLMs to comprehend image tables and assess the comparative performance of LLMs with text tables versus VLMs with image tables. Their findings indicate that representing tables in image form can facilitate complex reasoning for VLMs. To process these tables, AI systems require tools that convert visual cues into visual and layout embeddings.

Preprocessing Some works like LayoutLM [56,57] leverage pre-built optical character recognition (OCR) tools and PDF parsers for preprocessing images and PDF files. Other research in multimodal table understanding involves converting tables into images. For instance, Table-LLaVA [13] converts HTML Web tables into images to augment training data, enabling the model to understand tables within documents better. Xia et al. [58] transform spreadsheets into images to explore the capabilities of VLMs in comprehending spreadsheets.

Visual embedding Visual embedding is the combination of image, position, and segment embeddings. To handle table images in documents, AI systems must encode images into features. For example, LayoutLM and TableVLM [46] utilize ResNet [59] to convert images into visual embeddings, whereas Table-LLaVA [13] employs a Vision Transformer (ViT) [60].

Layout embedding Layout embeddings capture the spatial layout information present in table images. Both LayoutLM and TableVLM normalize and discretize coordinates to integer values within the range $[0, 1000]$, employing separate embedding layers for the x and y axes to represent the two-dimensional features distinctly.

4 Table training

In this section, we explore the training techniques of large models for table tasks. There are primarily two types of LLMs: large language models (LLMs) that accept only text input and visual language models (VLMs) that can process visual inputs. Given the differences in inputs, model architecture, and training techniques between these two types, we summarize table training techniques in Section 4.2.1 and will discuss each category individually. We begin by reviewing the literature on small language models (SLMs) prior to the LLM era, where these studies train models with fewer than a billion parameters. We will then delve into the details of LLMs and VLMs for tables.

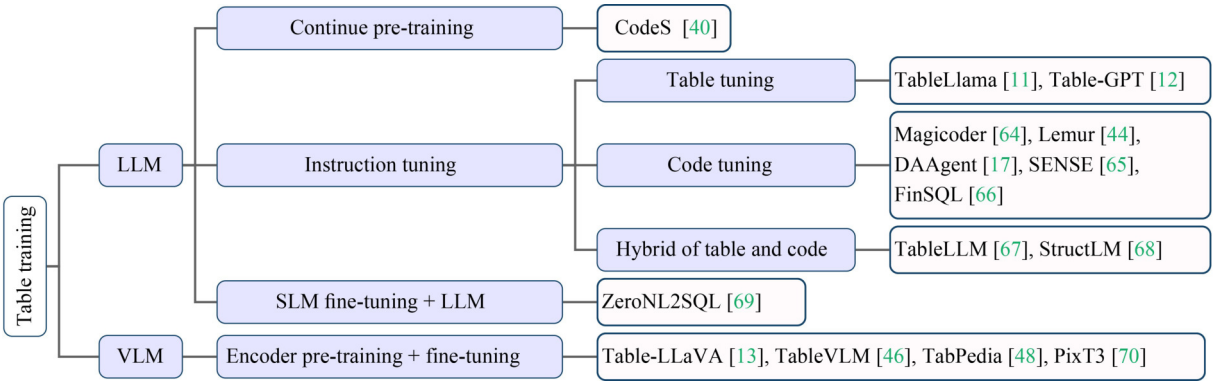


Fig. 5 Summary of Table LLM training techniques

4.1 Pre-LLM era

Before the era of LLMs, many researchers were already employing language models to address table tasks. Their works primarily focus on modifying model structures, devising encoding methods, and designing training objectives to tailor the models for table tasks. For example, TaPas [8] extends BERT's [61] model architecture and mask language modeling objective to pre-train and fine-tune with tables and related text segments. TaBERT [7] encodes a subset of table content most relevant to the input utterance and employs a vertical attention mechanism. TURL [9] encodes information of table components (e.g., caption, headers, and cells) into separate input embeddings and fuses them together. TABBIE [62] modifies the training objective to detect corrupted cells. TaPEx [10] learns a synthetic corpus, which is obtained by automatically synthesizing executable SQL queries and executing these SQLs. RESDSQL [63] injects the most relevant schema items into the model during training and ranks schema items during inference to find the optimal one.

To process image tables and tables within documents, researchers often adopt the encoder-decoder architecture. In this architecture, the encoder encodes visual information while the decoder generates textual output. For instance, LayoutLM [56,57] integrates textual, visual, and layout embeddings into its BERT backbone.

However, the foundation models of these methods are relatively small. Some cannot adapt to various downstream tasks, and some require annotated data during fine-tuning.

4.2 Table LLM training

4.2.1 What's new in table LLM training

Methods such as instruction tuning and continued pre-training are widely utilized in the era of LLMs. Although these approaches have already been employed in the pre-LLM era, their training techniques differ from those used previously.

Instruction tuning involves directing the model with specific instructions or guidance within the prompt. It enables the language model to follow the instructions given in the prompt. As shown in Fig. 5, for table processing tasks, there are three types of instruction tuning: table tuning, code tuning, and a hybrid of both table and code.

- **Table tuning** focuses on LLMs' understanding of tables so that LLMs can handle various table tasks such as table QA, table-to-text, entity linking, etc. This type of research utilizes general-purpose foundation LLMs (e.g., Llama) and a substantial volume of table-related data for instruction tuning. Table tuning examples include TableLlama [11] and Table-GPT [12].
- **Code tuning** addresses table processing tasks from a code generation perspective by generating code (e.g., SQL or Python) to manipulate table data. Code tuning examples include Magicoder [64], Lemur [44], and DAAgent [17].
- **Hybrid of table and code** research reveals that table instruction tuning based on code LLMs is more effective, i.e., code LLMs are tuned using table instruction datasets. For example, TableLLM [67] and StructLM [68] tune code LLMs on table datasets.

Building Instruction Datasets is of great importance for high-quality instruction tuning. Manually crafting these datasets is highly time-consuming and labor-intensive, presenting a major challenge in constructing high-quality datasets. So research focuses on constructing datasets via automatic approaches like using templates to transform existing annotated datasets for instruction tuning or distilling data from more powerful LLMs. Therefore, researchers concentrate on constructing instruction datasets automatically, employing methods such as template-based transformation of existing annotated datasets, or data distillation from more powerful LLMs.

Continue pre-training takes an existing trained model, feeds new data, and adapts it for a specific task [71]. **SLMs + LLMs** is a paradigm to fine-tune an SLM to guide the LLM towards desired outputs where two types of models complement each other [72].

4.2.2 Table tuning

Table tuning, short for table instruction tuning, constructs instruction tuning datasets by leveraging multiple existing table-related datasets. The instruction tuning dataset can be in the form of (*Instruction*, *Table*, *Output*). Figure 6 is an example entry from TableInstruct, the instruction tuning dataset for TableLlama [11]. In this example, the *Instruction* specifies the task; the *Table* describes table content, table metadata, or task-specific content. The *Output* features natural language outputs like table QA answers, text from table-to-text transformations, result tables after manipulation, or a mix of text and tables. This example is about fact verification and

An example entry from `TableInstruct`

Instruction:
This is a table fact verification task. The goal of this task is to distinguish whether the given statement is entailed or refuted by the given table.
Table:
[TLE] The table caption is about tony lema. [TAB] | tournament | wins | top - 5 | top - 10 | top - 25 | events | cuts made [SEP] | masters tournament | 0 | 1 | 2 | 4 | 4 | 4 | [SEP] ...
Question:
The statement is:<tony lema be in the top 5 for the master tournament, the us open, and the open championship>. Is it entailed or refuted by the table above?
Output:
Entailed.

Fig. 6 An example entry from `TableInstruct`, a table instruction tuning dataset

has a NL question. For other table tasks, the *Question* element may be optional. When using Web tables, titles or captions are included in the table content to provide context information to LLMs.

TableLlama emphasizes using more realistic data and uses the *template* approach to collect 14 existing datasets (e.g., WikiTableQuestions [32] or Spider [42]) of 11 table tasks. Table-GPT employs the *synthesis-then-augment* approach. The synthesis-then-augment approach resembles the method used in computer vision, where images are randomly cropped or flipped to create variations. Table-GPT designs 18 synthesis processes, ranging from table QA to row/column swapping. For example, the row/column swapping synthesis process swaps rows or columns, and the *Output* is the swapped table. In this way, the model can understand the order of rows/columns. Additionally, Table-GPT implements augmentation strategies at the instruction level, table level, and output level to increase task and data diversity. For example, the instruction level augmentation uses powerful LLM to paraphrase the canonical human-written instruction into many different variants. These augmentation approaches can prevent the model from overfitting. TableLlama and Table-GPT demonstrate that after table tuning on seen table tasks, LLMs could exhibit robust generalization capabilities and tackle unseen table tasks.

4.2.3 Code tuing

Tables can be manipulated through programming languages such as Python or SQL. Consequently, the code instruction tuning approach develops LLMs specializing in code generation. Models tuned with table instructions typically generate table-related content directly. Code LLMs first generate code that is subsequently executed in environments like Python interpreters or database engines. Code LLMs are particularly adept at tasks like NL2SQL and data analysis. Several code LLMs achieve top rankings on the DS-1000 [45] and InfiAgent-DABench [17] leaderboards, which are benchmarks for data analysis code generation. Here, we list a few examples and discuss how they build instruction datasets.

A sample prompt from `Evol-Instruct` for code

Please increase the difficulty of the given programming test question a bit. You can increase the difficulty using, but not limited to, the following methods:
Method:
Provide a piece of erroneous code as a reference to increase misdirection.
Problem:
{problem}

Fig. 7 A sample prompt from `Evol-Instruct` for code

WizardCoder [73] employs the "Evol-Instruct" method, where "Evol" denotes evolution, indicating the use of existing instruction data as a seed to prompt a more powerful LLM to generate new instructions. Figure 7 shows a sample prompt from Evol-Instruct for code. `Problem` refers to the current code instruction awaiting evolution and `Method` is evolution type. WizardCoder uses five heuristic evolution methods, and here in Fig. 7, we provide one example heuristic method.

Magicoder [64] propose a process called "OSS-Instruct". OSS-Instruct first collects open-sourced code snippets and lets a powerful LLM draw inspiration from the code snippets to produce realistic code instructions. OSS-Instruct is orthogonal to existing data generation methods like Evol-Instruct. Thus, these two methods can be combined together. Both WizardCoder and Magicoder utilize powerful LLMs to distill and generate additional data. Lemur [44] argues that code LLM should balance general-purpose ability with code ability. The general-purpose ability is for reasoning and planning and can be learned by NL text. The code ability, which can be learned from code, ensures grounding in programming environments. So, the authors build a corpus with a 10:1 code-to-text ratio to ensure that the trained code LLMs have coding ability while maintaining performance in NL ability. Lemur also evaluates whether the model performs effectively with agents that heavily depend on tool usage and environment feedback. DAAgent [17] 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 keywords and questions by iteratively prompting GPT-4 given a specific crawled CSV file. These studies show that enhancing models with code instructions can boost performance on table tasks, especially on data analysis tasks. When constructing code instruction datasets, these methods more or less distill data from powerful LLMs (e.g., GPT).

SENSE [65] is an NL2SQL model that utilizes two types of synthetic training data: "strong data" and "weak data." "Strong data" is distilled from powerful LLMs that provide more reliable responses, while "weak data" is produced by inferior

models that may result in errors during SQL execution. The strong and weak data are then trained with Direct Preference Optimization (DPO) [74], enabling the SENSE model to learn from both correct and incorrect samples. FinSQL [66] is also an NL2SQL solution. It creates a dataset specialized for the financial sector and employs a powerful LLM to augment the data, followed by parameter-efficient fine-tuning (PEFT) [75,76] of a LLM.

4.2.4 Hybrid of table and code

Currently, there are many open-source general-purpose models and code LLMs available. One question that arises is which foundation model should be selected for further training specifically for table tasks. StructLM [68] performs an ablation study using code LLM, general LLM, and math LLM as foundational models, fine-tuning them on tabular datasets. Studies [67,77], including StructLM reveal that using the code LLM as the foundation model achieves superior performance on table tasks. Like table instruction tuning, studies of the hybrid type focus on how to construct table instruction datasets.

4.2.5 Continue pre-training

Small-sized LLMs have lower deployment costs, but their code generation or reasoning abilities are inferior to those of large-sized LLMs. CodeS [40] proposes continued pre-training [71] on small-sized LLMs to enhance their performance in NL2SQL tasks. Specifically, CodeS feeds SQL-related, NL text, and NL-to-code data into the pre-trained StarCoder models [78], thereby enhancing the models' capabilities in natural language processing, reasoning, and coding.

4.2.6 SLMs + LLMs

SLMs are easier to fine-tune to understand table schemas, whereas LLMs exhibit strong reasoning capabilities but may encounter "hallucination" [79] problems owing to the lack of domain knowledge. ZeroNL2SQL [69] combines the strengths of both SLMs and LLMs to mitigate their respective weaknesses. It fine-tunes an Encoder-Decoder SLM responsible for generating SQL sketch candidates, and it employs an LLM to fill in missing parts in the SQL sketch, correct errors in the SQL query, and generate the final query.

4.3 Table VLM training

Research in utilizing VLMs for table tasks can be divided into three types. The first type adheres to the conventional pattern recognition method [56,80], which involves table detection and extraction tasks. A notable example is the TableVLM [46]. The second approach tackles various table tasks in an end-to-end manner. The example is Table-LLaVA [13]. The third type integrates features of the first two, enabling the detection, extraction, and end-to-end tasks such as question answering on image tables, with TabPedia [48] serving as an example. The third type is a hybrid of the first two, which can detect and extract tables and answer questions based on table images, exemplified by TabPedia [48].

Pre-training + fine-tuning Most LLMs are designed with a decoder-only architecture. Similar to the network architecture of visual language models from the pre-LLM era, VLMs typically employ an encoder-decoder architecture, where a visual encoder converts visual data into embeddings while the decoder generates texts. The encoder may utilize architectures such as ResNet or ViT, while the decoder typically consists of a pre-trained LLM. Compared to the decoder, which has been thoroughly trained, the encoder has not yet mastered much visual information about tables. It is necessary to align the visual cues with the textual information. Therefore, the training process for a table VLM is usually divided into two phases: 1) Pre-training the visual encoder while freezing the parameters of the LLM decoder, and 2) fine-tuning or instruction tuning the entire model. Both phases require a substantial amount of high-quality training data. PixT3 [70] is a multimodal table-to-text model that takes table-to-text tasks as table visual recognition tasks and generates texts, removing the need to process tables in text formats.

5 Table prompting

In this section, we explore the strategies for prompting large models to handle table tasks. LLMs struggle with functionalities, such as complex reasoning, arithmetic calculation, factual lookups, and correcting erroneous decisions, all essential for table tasks. Thus, the key challenges include guiding the model towards complex reasoning, enabling it to reflect and revise rather than fast thinking, and utilizing external tools for executing Python or SQL code. To address the issues above, researchers have been dedicated to developing LLM-powered agents.

5.1 Common workflow of LLM-powered agents

Research such as Chain-of-Thought (CoT) [20] and ReAct [81] prompt LLMs iteratively, organizing the reasoning process into multiple intermediate steps. Thus, LLMs address simpler subproblems step by step and progressively build a coherent response. Agent systems are designed to follow this paradigm and typically include modules such as **memory**, **planning**, and **action** [21]. The memory module stores state or observations from the environment or records past actions. This information can be utilized for future planning. The planning module chooses which action the agent needs to do in the current step, while the action module interacts with the environment and executes the action to get the outcomes. As shown in Fig. 8, for table tasks, the agent first observes the table data and user intent. It generates prompts, decomposes complex tasks, plans actions, executes them on the table environment, and then updates the state or observation. This iterative process is repeated until expectations are met. Studies like SheetAgent [38], Chain-of-Table [31], ReAcTable [15], TAPERA [82], and E^5 [83] follow this workflow.

The memory module enables the agent to accumulate experiences, self-evolve, and act with greater consistency, rationality, and effectiveness [21]. The memory of table agents stores planning history, that is, the outcomes of specific actions and table states, typically in a structured table format. In other aspects, the memory module of table agents is not significantly different from that of other LLM-based agents. This paper focuses on the planning and action modules of table agents.

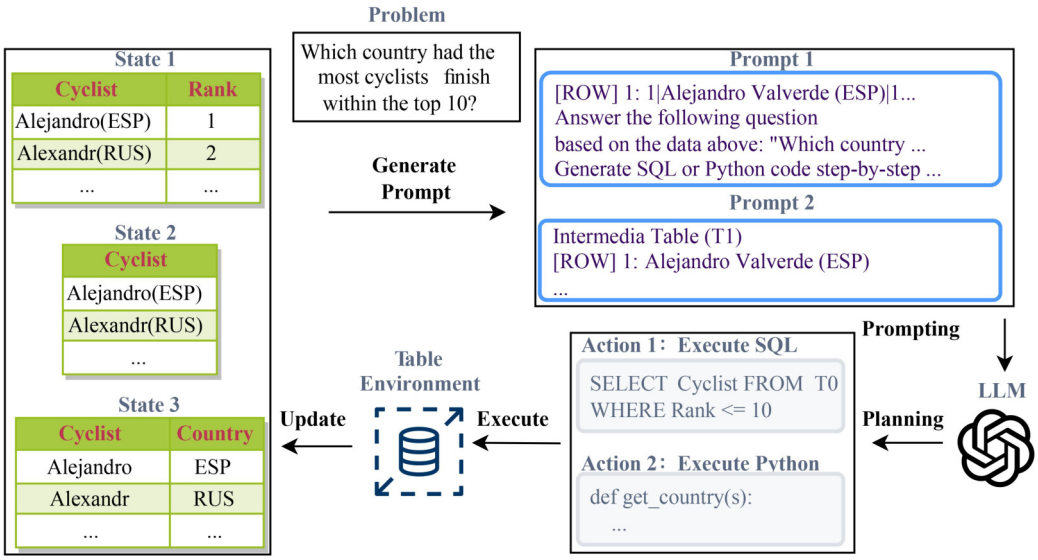


Fig. 8 The iterative process of an LLM-powered agent system for table tasks

5.2 Planning

The planning module plans actions by prompting LLMs. It must carefully handle two key aspects: 1) breaking down complex problems into smaller sub-problems, and 2) reflecting on and revising previous decisions.

5.2.1 Formalizing planning

In table agents, the planning module interacts with the target tables using a ReAct approach, with feedback and reflection. SheetAgent [38] provides a formal definition of planning for spreadsheet manipulation, and here, we further extend it to encompass more table tasks like table QA and NL2SQL. Typically, the input of the planning module usually consists of the task instruction I, table state S_t at step t, user query or current (sub-)problem Q, planning history H_{t-1}. Utilizing this information, the planning module prompts LLMs and formulates an action A_t for step t by:

$$A_t = P(A_t|I, S_t, Q, H_{t-1}).$$

The action, which we will discuss in Section 5.3, is executed on the target table, yielding a new observation and the output denoted as O_t. The table state and the planning history are then updated by $H_t = (H_{t-1}, O_t, A_t)$.

5.2.2 Complex task decomposition

Inspired by the CoT and least-to-most [84] prompting methods, researchers instruct LLMs to decompose complex table tasks into simpler sub-tasks. DIN-SQL [85] proposes breaking down the NL2SQL task into subtasks. The process involves identifying the relevant tables and columns associated with the query and producing intermediate sub-queries. DEA-SQL [86] and TabSQLify [87] also follow the principle of decomposing tables into smaller, simplifying subtasks and reducing irrelevant information. For Web tables, Dater [33] exploits 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.

5.2.3 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 The *self-consistency* [88] decoding strategy plans multiple reasoning paths and chooses the most consistent answer. It can significantly enhance LLM's accuracy on complex tasks. This has also been demonstrated in table tasks, which are reported in papers of Dater [33], DAIL-SQL [41], MCS-SQL [89], and ReAcTable [15] on table QA and NL2SQL tasks. However, several studies [15,41] argue that while self-consistency and voting enhance accuracy, these methods are time-consuming, and the cost is higher, considering that prompting LLMs is not cheap.

Revising In many table tasks, a mistake in one step's action can significantly impact the following analysis and processing. To handle this challenge, SheetCopilot [16] and SheetAgent [38] adopt a mechanism that reflects and refines past actions. SheetAgent features a Retriever component that retrieves high-quality code examples from their curated repository, which helps to prevent the generation of erroneous code. SelfEvolve [90] 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. These techniques prevent error actions like incorrect APIs or wrong arguments.

5.3 Action

In table tasks, actions are intermediaries between LLMs and software tools like database engines, spreadsheet systems, or Python interpreters. Utilizing these external tools involves more than just converting NL text into software APIs. The agent system must ensure that the API calls are error-free and can address complex tasks. These external tools extract data from tables and are similar to the Retrieval-Augmented Generation (RAG) [91] concept. Given the inherent characteristics of different table tasks, we will discuss how to define actions based on the specific tasks.

5.3.1 Table QA and NL2SQL

For tasks like table QA and NL2SQL, agent systems usually utilize Python or SQL to interact with tables. In Binder [30], actions are categorized into two types: extended Python and SQL, which allow the injection of LLM as operators within standard Python and SQL. ReAcTable [15] 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.

5.3.2 Spreadsheet manipulation and data analysis

Spreadsheet manipulation and data analysis are highly flexible. First, users' expectations and requirements vary. Second, there is a wide range of operations and software APIs. Third, a complex task may contain multiple operations, leading to dynamic alterations in the table content.

SheetCopilot [16], an agent system for spreadsheets, models existing spreadsheet VBA APIs into atomic actions. An atomic action comprises the API name, argument lists, document string, and several usage examples. These atomic actions are not specific to one system and can be implemented on different spreadsheet systems. The authors design these atomic actions by crawling spreadsheet-related questions online, embedding and clustering them into categories, and abstracting them to VBA APIs by choosing the most representative ones. SheetAgent [38] finds that Python and SQL are more appropriate for spreadsheet manipulation than VBA, owing to the two programming languages being more aligned with the training data of existing LLMs. SheetAgent comprises two essential components for agent actions: a Planner and an Informer. The Planner generates Python code and employs a ReAct-style reasoning approach to manipulate the spreadsheet table. The Informer serves as an evidence provider, supplying subtask-specific SQL queries to assist the Planner in tackling complex tasks. Data-Copilot [43] 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. When designing interfaces, Data-Copilot generates code for each user request. It then assesses whether there are similarities among these requests that can be merged or abstracted.

5.3.3 Multiple table tasks

Most prompting methods are for a single table task; here, we list several studies that address multiple tasks. These frameworks share common characteristics: they first explore the data, including the data itself and its schema, and then plan and optimize actions.

StructGPT [92] is capable of solving table QA, NL2SQL, and knowledge graph QA 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. UniDM [36] manages a variety of data manipulation tasks, such as data cleaning, on data lakes through a three-action process. The first action is extracting context information, including table metadata and related records associated with the task, utilizing this information as demonstrations or background knowledge. The second action involves converting this context information into NL texts suitable for LLMs to reason. The final action employs prompt engineering to formulate the desired prompt. TAP4LLM [93] is a pre-processing toolbox to generate table prompts. Its action includes: 1) selecting appropriate rows and columns from the table (called *Table Sampling*), 2) integrating them with other table data (relevant external knowledge, metadata) (called *Table Augmentation*), and 3) serializing this information to fit the context length of the LLM. ChatPipe [94] is a system designed to facilitate effortless interaction between users and ChatGPT for table data analysis tasks. Initially, users upload their dataset along with their query. ChatPipe assists in analyzing the dataset and suggests various data analysis and feature engineering operations. Given the numerous potential data operations, ChatPipe employs a reinforcement learning-based method, utilizing a Deep Q-Network model [95] to learn and recommend the most suitable operations.

5.4 Other techniques

This subsection discusses other prompting techniques like few-shot examples of in-context learning [19] and role-play.

5.4.1 Few-shot examples selection

In-context learning refers to the ability of models to learn from examples within the context of prompts, an ability that emerges in LLMs. The key to in-context learning is how to select and organize the most helpful demonstrative examples into the prompt. For the NL2SQL task, selecting the most relevant example can be achieved by choosing example queries more related to the target NL question or example SQL more similar to the potential SQL. Nan et al. [54] argue that diversity should be considered in addition to similarity. As the context length is limited, organizing all the example information (i.e., NL question, schema, SQL) into the prompt ensures the quality of examples but sometimes may exceed the context length. DAIL-SQL [41] proposes a method that balances the quality and token quantity by removing examples' schemas, which are token-cost.

5.4.2 Role-play

LLMs can be assigned roles when prompting them. For instance, Zhao et al. [96] test a prompt like "Suppose you are an expert in statistical analysis". Tapilot-Crossing [97] utilizes a multi-agent environment to generate user intents and simulate use cases from real-world scenarios. Within this environment, each agent takes on a unique role, such as Administrator, Client, Data Scientist, or AI Chatbot, interacting with each other to mimic a realistic data analysis context.

6 Resources

In this section, we summarize open-source datasets, benchmarks, and software, as these artifacts can facilitate the community's progress.

6.1 Datasets and benchmarks

Traditional benchmarks, such as WikiTQ [32], WikiSQL [98], and Spider [42], 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 data sources, sizes. 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 [103] and RobuT [99] are proposed. RobuT reveals that the performance of all table methods degrades when perturbations are introduced, yet close-source LLMs (e.g., GPT) exhibit greater robustness.
- **Human involved labeling** Some new datasets, which target data analysis, require extensive manual annotation. DS-1000 [45], InfiAgent-DABench [17], Tapilot-Crossing [97] are designed to assess data analysis tasks. They gather data from the internet and annotate it either automatically or semi-automatically. DS-1000 collects questions from StackOverflow, assesses their usefulness manually, and curates to form the benchmark. The authors manually adapt the original questions by providing input and output context into test cases and rewriting problems to prevent LLMs from learning and memorizing the data. InfiAgent-DABench invites human experts to evaluate the dataset quality and compare human-made and GPT-4 generated data analysis questions via multiple metrics. Tapilot-Crossing aims to build a benchmark for real-world data analysis. An issue that cannot be overlooked is the quality of these datasets. Wretblad et al. conduct a thorough analysis of the NL2SQL dataset BIRD [101], discovering inaccuracies in some of the gold SQLs and noise within certain NL queries [109]. The creators of the BIRD dataset introduce this noise during the dataset creation process. After correcting these errors, Wretblad et al. [109] find that complex prompting methods (e.g., DIN-SQL) might be less effective than simple zero-shot prompting. This research reveals the potential unreliability of table datasets such as BIRD, considering they are generated through human labeling, a process prone to introducing noise and errors.
- **Real-world workload** Most datasets are derived from the Web or synthesized using templates. These publicly available online data are usually simple, as some of them are just tutorials for beginners. In contrast, tables in real-world scenarios are far more complex than these. ScienceBenchmark [104] introduces a real-world benchmark developed in collaboration with SQL experts and researchers specializing in policy-making, astrophysics, and cancer research. Given the scarcity of real-world data in these domain-specific databases, ScienceBenchmark employs a data augmentation strategy that starts with hundreds of human-labeled NL2SQL pairs to create thousands more data points. SpreadsheetBench [39] proposes a benchmark aimed at real-world scenarios, where the authors meticulously analyzed user questions from four Excel forums. They argue that their benchmark could assess the performance of handling complex user instructions.
- **Larger scale** AnaMeta [105] is a large-scale table metadata dataset. GitTables [107] downloads millions of CSV tables from GitHub and aligns them with knowledge bases. SchemaPile [108] is a corpus with 221k database schemas and 1.7 million table definitions. These datasets can help evaluate whether the AI system could understand the table schema and benchmark tasks like column type annotation. They can also be fed into neural models as training data for solving various downstream tasks. ComplexTable [46] contains more than 1 million image tables featuring complex structures and can be utilized for visual tasks related to tables.

Table 2 Datasets and benchmarks for table processing tasks

Dataset	Table task	Data sources	Size
RobuT [99]	Table QA	WikiTQ [32], WikiSQL [98], SQA [100]	138,149 perturbed examples
BIRD [101]	NL2SQL	kaggle.com, CTU Prague [102] Open tables,	81 DBs, 12,751 NL2SQL pairs
Dr.Spider [103]	NL2SQL	Spider [42]	200 DBs, 15k perturbed examples
ScienceBenchmark [104]	NL2SQL	Human + AI Augmented	3 DBs, 6k NL2SQL pairs
SheetCopilot [16]	Spreadsheet manipulation	superuser.com	28 SSs, 13k QA pairs
SpreadsheetBench [39]	Spreadsheet manipulation	4 Excel online forums, e.g., excelfourm.com	912 instructions, 2,729 test cases
DS-1000 [45]	Data analysis	stackoverflow.com	451 problems
InfiAgent-DABench [17]	Data analysis	github.com	631 CSVs, 5131 samples
Tapilot-Crossing [97]	Data analysis	kaggle.com	1176 user intents
AnaMeta [105]	Entity linking, Column type annotation	public Web, TURL [9], SemTab [106]	467k WT/SSs
GitTables [107]	Column type annotation, Column population	Github.com	1M CSVs
SchemaPile [108]	Column type annotation, Column population NL2SQL Data analysis	Github.com	221k DB schemas 1.7M table definitions
ComplexTable [46]	Table detection, Table extraction	Synthetically generated	1M tables (png, HTML)

6.2 Open-source software

The academia and industry have developed numerous open-source software; here, we select a few to discuss. LlamaIndex [110] is a popular and versatile RAG framework that, while supporting some table tasks, is not specialized in this area. Both DB-GPT [111] and Vanna [112] are AI tools designed for database interactions. They allow users to train models or utilize their built-in prompting features. PandasAI [113] enables users to clean, query, and visualize pandas DataFrames using NL questions. Most of these tools are designed for NL2SQL and table QA tasks, which enable users to query tables with NL texts. This trend underscores the demand among general users to utilize NL to enhance the efficiency of table processing. RetClean [37] is a tool that leverages LLMs for data cleaning operations, such as missing value imputation.

7 Analysis and discussion

In order to demonstrate the accuracy and costs of various methods to the readers, this section presents a comparative analysis and discusses the advantages and disadvantages of different approaches. Specifically, utilizing data from several papers [17,39,44,67,77,114], we summarize both accuracy and cost for four table tasks: table QA, NL2SQL, spreadsheet manipulation, and data analysis.

7.1 Discussion on LLM training

The advantages of training-based methods are that they offer great control, allowing enterprises to conduct private training and deployment without data leakage to third-party model service providers. However, two issues can not be ignored: cost and accuracy.

Cost The expense of pre-training or fine-tuning a large model is substantial. Fine-tuning a 7B model typically requires eight 80 GB GPUs, and the number of training tokens dictates the total time needed for training. The training speed, measured in tokens per second, is affected by both the hardware and the software and is reported in several technical reports, such as Llama [115]. Table 3 lists the number of parameters and training tokens for two LLM training approaches, thereby providing readers with an understanding of the training costs. The number of parameters can also be used to estimate the minimal costs for private deployment. For instance, loading a 7B model with float16 format requires at least 14 GB of memory, while serving with concurrent requests necessitates additional memory for the transformer architecture's key-value (KV) cache [116].

Accuracy As indicated in Table 3, across four benchmarks, the LLM training approach excels in only one, whereas prompting a robust LLM (GPT-4) is superior. The 7B TableLlama can't outperform task-specific fine-tuning models on specific tasks (see the WiKiSQL column in Table 3). Another example is the NL2SQL task, where Li et al. [77] perform a systematic evaluation with various methods, comparing LLM-based and SLM-based solutions. Results show that there is no clear winner between LLM and SLM solutions on different metrics and domains. On data analysis tasks, instruction tuned models like Lemur [44] and DAAgent [17] cannot surpass GPT-4. Table-GPT continues to train on GPT-3.5 and achieves 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 distillation. Evidently, GPT is the performance ceiling.

7.2 Discussion on LLM prompting

The LLM-powered agent method combines the power of LLMs and the flexibility of external tools. On many table tasks, prompting strong LLMs like GPT still performs the best.

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 As shown in the "Avg. # of Infers" column of Table 3, agents must prompt LLMs repeatedly to fulfill users' expectations. Ma et al. [39] conduct experiments on agents for spreadsheet manipulation, and the most effective agent solution is the one that utilizes the ReAct framework [81], executes code within the execution environment and offers feedback to LLMs upon failure. Deploying multiple rounds of

Table 3 A comparative analysis of various methodologies was conducted using four benchmarks in table QA and NL2SQL tasks. The experimental setup and performance metrics are referenced from corresponding papers. The number of parameters and training tokens are derived from the respective papers detailing each method

Type	Method	# of parameters	# of tokens to train	Benchmarks				Avg. # of infers
				WikiTQ	FeTaQA	WikiSQL	Spider	
SLM training	TaPEX	0.14B	–	38.55	–	83.90	15.04	1
	TaPas	0.11B	–	31.60	–	74.20	23.05	1
LLM training	TableLlama	7B	3.3B	48.82	67.73	43.70	–	1
	TableLLM	13B	1.1B	62.40	74.50	**90.70**	83.40	1
Prompting	CodeLlama	13B	–	43,44	57.24	38.30	21.88	1
	GPT-3.5	–	–	58.45	71.18	81.70	67.38	1
	GPT-4	–	–	**74.09**	**78.35**	84.00	69.53	1
Prompting agent	StructGPT(GPT-3.5)	–	–	52.45	11.80	67.80	**84.80**	3
	Binder(GPT-3.5)	–	–	61.61	12.77	78.60	52.55	50
	DATER(GPT-3.5)	–	–	53.40	18.26	58.20	26.52	100

ReAct prompting with execution feedback takes a substantial time. Consequently, all these factors increase the total 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. As open-source models advance, enterprises can also deploy open-source models or their own fine-tuned versions, thus gradually mitigating this issue.

Structure understanding Foundation models, without fine-tuning for tables, still have difficulty understanding table structure and hierarchy. Pang et al. [114] create a benchmark named TIS to assess how effectively LLMs seek information from tables. They discover that LLMs struggle with tables that have complex hierarchies and exhibit a poor understanding of table structures, such as locating a specific regin in a two-dimensional table. Notably, most LLMs perform at nearly random accuracy levels (around 50%) in the table structure understanding task, while GPT-4 achieve 66.1%.

8 Challenges and future directions

In this section, we outline some challenges and considerations for future research.

8.1 Diverse user input when serving

User inputs refer not only to users' NL queries but also to the table schema and content.

User query In real-world applications, user NL queries are frequently ambiguous. For example, when users are unfamiliar with the table they want to query, they often pose general requests without a clear objective, such as "help me analyze this table". Users might also pose questions unrelated to the table within a table AI system. Even when users have some knowledge of the table schema and content, they may struggle to formulate their query accurately, leading to ambiguity in the query sentences.

Table schema and content In practice, table schemas and content are highly diverse and often proprietary. For instance, in a domain or industry with which the model has limited knowledge, it faces challenges in effectively transferring into the field. Existing table training datasets are relatively simple, either consisting of simple table structures scraped from the Web or synthesized based on powerful LLMs. Constructing a complex and diverse training dataset is quite costly. Ensuring that the training data covers real-world business scenarios poses a significant challenge.

Future table LLMs should adapt quickly and cheaply to real-world business needs. Research directions include synthesizing high-quality training data that reflects the diverse needs of specific domains by cost-effective methods.

8.2 Slow and deep thinking

Daniel Kahneman has revealed that slow and deep thinking is the underlying mechanism of the human brain for processing complex problems [117]. The chain-of-thought approach is considered to guide LLMs in solving complex reasoning problems, and the recent OpenAI's o1 shows that LLMs can achieve substantial enhancements in performance by allowing them to generate extended internal chains-of-thought. Additionally, this internal chain-of-thought can facilitate both the model training processes and the pure model prompting methods.

Training Typically, there is a considerable gap between the user instructions and the ultimate answer. Therefore, a chain-of-thought can serve as an internal mechanism to explain the process of deriving the final answer, potentially mitigating the training difficulty faced by LLMs. However, acquiring an accurate chain-of-thought is challenging. It can be prohibitively expensive when achieved through human annotation and difficult to ensure accuracy when automatically generated by LLMs themselves, particularly in tasks involving complex logical reasoning over tables. Exploring cost-effective methods to guarantee an accurate chain-of-thought process is a valuable area of research.

Prompting In existing prompting methods on table tasks, researchers develop their own chain-of-thought workflows, which heavily rely on hard-coded prompt templates to solve specific table tasks. This results in methods with weak transferability. Given that OpenAI's o1 possesses inherent chain-of-thought abilities, prompting and agent methods must reconsider how to construct their workflows. On the other hand, the chain-of-thought approach requires numerous inference iterations, which is time-consuming; future table prompting methods should balance inference times with accuracy.

9 Conclusion

This survey is the first comprehensive investigation into Large Language Models (LLMs) for table processing across various tasks, encompassing table QA, spreadsheet manipulation, data analysis, etc. We provide a summary and categorization of table tasks from both academic and end-user perspectives. We explore the popular and essential techniques for table processing, including data representation, training, and prompting. We collect and discuss resources like open-source datasets, benchmarks, and software. Beyond reviewing existing work, we also identify several challenges within this domain that could inform and guide future research directions.

Acknowledgements This work was supported by the National Key R&D Program of China (2023YFF0725100), the National Natural Science Foundation of China (Grant Nos. 62322214, 62272466), and the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China (24XNKJ22).

Competing interests The authors declare that they have no competing interests or financial conflicts to disclose.

References

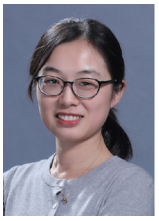
1. Dong H, Cheng Z, He X, Zhou M, Zhou A, Zhou F, Liu A, Han S, Zhang D. Table pre-training: a survey on model architectures, pre-training objectives, and downstream tasks. In: Proceedings of the 31st International Joint Conference on Artificial Intelligence. 2022, 5426−5435
2. Badaro G, Saeed M, Papotti P. Transformers for tabular data representation: a survey of models and applications. Transactions of the Association for Computational Linguistics, 2023, 11: 227−249
3. Fang X, Xu W, Tan F A, Zhang J, Hu Z, Qi Y, Nickleach S, Socolinsky D, Sengamedu S, Faloutsos C. Large language models(LLMs) on tabular data: prediction, generation, and understanding — a survey. 2024, arXiv preprint arXiv: 2402.17944
4. Zhang X, Wang D, Dou L, Zhu Q, Che W. A survey of table reasoning with large language models. 2024, arXiv preprint arXiv: 2402.08259
5. Zhao W X, Zhou K, Li J, Tang T, Wang X, Hou Y, Min Y, Zhang B, Zhang J, Dong Z, Du Y, Yang C, Chen Y, Chen Z, Jiang J, Ren R, Li Y, Tang X, Liu Z, Liu P, Nie J Y, Wen J R. A survey of large language models. 2023, arXiv preprint arXiv: 2303.18223
6. Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou Y, Li W, Liu P J. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 2020, 21(1): 140
7. Yin P, Neubig G, Yih W, Riedel S. TaBERT: pretraining for joint understanding of textual and tabular data. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020, 8413−8426
8. Herzig J, Nowak P K, Müller T, Piccinno F, Eisenschlos J. TaPas: weakly supervised table parsing via pre-training. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020, 4320−4333
9. Deng X, Sun H, Lees A, Wu Y, Yu C. TURL: table understanding through representation learning. Proceedings of the VLDB Endowment, 2020, 14(3): 307−319
10. Liu Q, Chen B, Guo J, Ziyadi M, Lin Z, Chen W, Lou J. TAPEX: table pre-training via learning a neural SQL executor. In: Proceedings of the 10th International Conference on Learning Representations. 2022
11. Zhang T, Yue X, Li Y, Sun H. TableLlama: towards open large generalist models for tables. In: Proceedings of 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2024, 6024−6044
12. Li P, He Y, Yashar D, Cui W, Ge S, Zhang H, Fainman D R, Zhang D, Chaudhuri S. Table-GPT: table fine-tuned GPT for diverse table tasks. Proceedings of the ACM on Management of Data, 2024, 2(3): 176
13. Zheng M, Feng X, Si Q, She Q, Lin Z, Jiang W, Wang W. Multimodal table understanding. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics. 2024, 9102−9124
14. Sui Y, Zhou M, Zhou M, Han S, Zhang D. 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. 2024, 645−654
15. Zhang Y, Henkel J, Floratou A, Cahoon J, Deep S, Patel J M. ReAcTable: enhancing ReAct for table question answering. Proceedings of the VLDB Endowment, 2024, 17(8): 1981−1994
16. Li H, Su J, Chen Y, Li Q, Zhang Z. SheetCopilot: bringing software productivity to the next level through large language models. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. 2023, 4952−4984
17. Hu X, Zhao Z, Wei S, Chai Z, Ma Q, Wang G, Wang X, Su J, Xu J, Zhu M, Cheng Y, Yuan J, Li J, Kuang K, Yang Y, Yang H, Wu F. InfiAgent-DABench: evaluating agents on data analysis tasks. In: Proceedings of the 41st International Conference on Machine Learning. 2024, 19544−19572
18. Wei J, Bosma M, Zhao V, Guu K, Yu A W, Lester B, Du N, Dai A M, Le Q V. Finetuned language models are zero-shot learners. In: Proceedings of the 10th International Conference on Learning Representations. 2022
19. Brown T B, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D M, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D. Language models are few-shot learners. In: Proceedings of the 34th International Conference on Neural Information Processing Systems. 2020, 159
20. Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, Chi E H, Le Q V, Zhou D. Chain-of-thought prompting elicits reasoning in large language models. In: Proceedings of the 36th International Conference on Neural Information Processing Systems. 2022, 24824−24837
21. Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, Chen Z, Tang J, Chen X, Lin Y, Zhao W X, Wei Z, Wen J. A survey on large language model based autonomous agents. Frontiers of Computer Science, 2024, 18(6): 186345
22. Lu W. Survey resources. See Github.com/godaai/llm-table-survey website, 2024
23. Jin N, Siebert J, Li D, Chen Q. A survey on table question answering: recent advances. In: Proceedings of the 7th China Conference on Knowledge Graph and Semantic Computing: Knowledge Graph Empowers the Digital Economy. 2022, 174−186
24. Qin B, Hui B, Wang L, Yang M, Li J, Li B, Geng R, Cao R, Sun J, Si L, Huang F, Li Y. A survey on text-to-SQL parsing: concepts, methods, and future directions. 2022, arXiv preprint arXiv: 2208.13629
25. Hong Z, Yuan Z, Zhang Q, Chen H, Dong J, Huang F, Huang X. Next-Generation database interfaces: a survey of LLM-based text-to-SQL. 2024, arXiv preprint arXiv: 2406.08426
26. Zhang S, Balog K. Web table extraction, retrieval, and augmentation: a survey. ACM Transactions on Intelligent Systems and Technology (TIST), 2020, 11(2): 13
27. Rahman S, Mack K, Bendre M, Zhang R, Karahalios K, Parameswaran A. Benchmarking spreadsheet systems. In: Proceedings of 2020 ACM SIGMOD International Conference on Management of Data. 2020, 1589−1599
28. Ritze D, Bizer C. Matching Web tables to DBpedia - a feature utility study. In: Proceedings of the 20th International Conference on Extending Database Technology. 2017, 210−221
29. Bhagavatula C S, Noraset T, Downey D. TabEL: entity linking in Web tables. In: Proceedings of the 14th International Semantic Web Conference on The Semantic Web - ISWC 2015. 2015, 425−441
30. Cheng Z, Xie T, Shi P, Li C, Nadkarni R, Hu Y, Xiong C, Radev D, Ostendorf M, Zettlemoyer L, Smith N A, Yu T. Binding language models in symbolic languages. In: Proceedings of the 11th International Conference on Learning Representations. 2023
31. Wang Z, Zhang H, Li C L, Eisenschlos J M, Perot V, Wang Z, Miculicich L, Fujii Y, Shang J, Lee C Y, Pfister T. Chain-of-table: evolving tables in the reasoning chain for table understanding. In: Proceedings of the 12th International Conference on Learning Representations. 2024
32. Pasupat P, Liang P. 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. 2015, 1470−1480
33. Ye Y, Hui B, Yang M, Li B, Huang F, Li Y. 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. 2023, 174−184

34. Chen W, Wang H, Chen J, Zhang Y, Wang H, Li S, Zhou X, Wang W Y. TabFact: a large-scale dataset for table-based fact Verification. In: Proceedings of the 33rd Neural Information Processing Systems. 2019
35. Parikh A, Wang X, Gehrmann S, Faruqui M, Dhingra B, Yang D, Das D. ToTTo: a controlled table-to-text generation dataset. In: Proceedings of 2020 Conference on Empirical Methods in Natural Language Processing. 2020, 1173−1186
36. Qian Y, He Y, Zhu R, Huang J, Ma Z, Wang H, Wang Y, Sun X, Lian D, Ding B, Zhou J. UniDM: a Unified framework for data manipulation with large language models. In: Proceedings of Machine Learning and Systems 6 (MLSys 2024) Conference. 2024
37. Ahmad M S, Naeem Z A, Eltabakh M, Ouzzani M, Tang N. RetClean: retrieval-based data cleaning using foundation models and data lakes. 2023, arXiv preprint arXiv: 2303.16909
38. Chen Y, Yuan Y, Zhang Z, Zheng Y, Liu J, Ni F, Hao J. SheetAgent: towards a generalist agent for spreadsheet reasoning and manipulation via large language models. 2024, arXiv preprint arXiv: 2403.03636
39. Ma Z, Zhang B, Zhang J, Yu J, Zhang X, Zhang X, Luo S, Wang X, Tang J. SpreadsheetBench: towards challenging real world spreadsheet manipulation. 2024, arXiv preprint arXiv: 2406.14991
40. Li H, Zhang J, Liu H, Fan J, Zhang X, Zhu J, Wei R, Pan H, Li C, Chen H. CodeS: towards building open-source language models for text-to-SQL. Proceedings of the ACM on Management of Data, 2024, 2(3): 127
41. Gao D, Wang H, Li Y, Sun X, Qian Y, Ding B, Zhou J. Text-to-SQL empowered by large language models: a benchmark evaluation. In: Proceedings of the VLDB Endowment, 2024, 17(5): 1132−1145
42. Yu T, Zhang R, Yang K, Yasunaga M, Wang D, Li Z, Ma J, Li I, Yao Q, Roman S, Zhang Z, Radev D. Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In: Proceedings of 2018 Conference on Empirical Methods in Natural Language Processing. 2018, 3911−3921
43. Zhang W, Shen Y, Lu W, Zhuang Y T. Data-Copilot: bridging billions of data and humans with autonomous workflow. 2023, arXiv preprint arXiv: 2306.07209
44. Xu Y, Su H, Xing C, Mi B, Liu Q, Shi W, Hui B, Zhou F, Liu Y, Xie T, Cheng Z, Zhao S, Kong L, Wang B, Xiong C, Yu T. Lemur: harmonizing natural language and code for language agents. In: Proceedings of the 12th International Conference on Learning Representations. 2024
45. Lai Y, Li C, Wang Y, Zhang T, Zhong R, Zettlemoyer L, Yih W T, Fried D, Wang S, Yu T. DS-1000: a natural and reliable benchmark for data science code generation. In: Proceedings of the 40th International Conference on Machine Learning. 2023, 18319−18345
46. Chen L, Huang C, Zheng X, Lin J, Huang X. TableVLM: multi-modal pre-training for table structure recognition. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics. 2023, 2437−2449
47. Li M, Cui L, Huang S, Wei F, Zhou M, Li Z. TableBank: table benchmark for Image-based table detection and recognition. In: Proceedings of the 12th Language Resources and Evaluation Conference. 2020, 1918−1925
48. Zhao W, Feng H, Liu Q, Tang J, Wei S, Wu B, Liao L, Ye Y, Liu H, Zhou W, Li H, Huang C. TabPedia: towards comprehensive visual table understanding with concept synergy. 2024, arXiv preprint arXiv: 2406.01326
49. Zhong X, ShafieiBavani E, Jimeno Yepes A. Image-based table recognition: data, model, and evaluation. In: Proceedings of the 16th European Conference on Computer Vision - ECCV 2020. 2020, 564−580
50. Abedjan Z, Chu X, Deng D, Fernandez R C, Ilyas I F, Ouzzani M, Papotti P, Stonebraker M, Tang N. Detecting data errors: where are we and what needs to be done? Proceedings of the VLDB Endowment, 2016, 9(12): 993−1004
51. Barke S, James M B, Polikarpova N. Grounded copilot: how programmers interact with code-generating models. Proceedings of the ACM on Programming Languages, 2023, 7(OOPSLA1): 78
52. Singha A, Cambronero J, Gulwani S, Le V, Parnin C. Tabular representation, noisy operators, and impacts on table structure understanding tasks in LLMs. In: Proceedings of the Table Representation Learning Workshop at NeurIPS 2023. 2023
53. Tian Y, Zhao J, Dong H, Xiong J, Xia S, Zhou M, Lin Y, Cambronero J, He Y, Han S, Zhang D. SpreadsheetLLM: encoding spreadsheets for large language models. 2024, arXiv preprint arXiv: 2407.09025
54. Nan L, Zhao Y, Zou W, Ri N, Tae J, Zhang E, Cohan A, Radev D. Enhancing text-to-SQL capabilities of large language models: a study on prompt design strategies. In: Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2023. 2023, 14935−14956
55. Deng N, Sun Z, He R, Sikka A, Chen Y, Ma L, Zhang Y, Mihalcea R. Tables as texts or images: evaluating the table reasoning ability of LLMs and MLLMs. In: Proceedings of the Findings of the Association for Computational Linguistics: ACL 2024. 2024, 407−426
56. Xu Y, Li M, Cui L, Huang S, Wei F, Zhou M. LayoutLM: pre-training of text and layout for document image understanding. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020, 1192−1200
57. Xu Y, Xu Y, Lv T, Cui L, Wei F, Wang G, Lu Y, Florencio D, Zhang C, Che W, Zhang M, Zhou L. LayoutLMv2: multi-modal pre-training for visually-rich document understanding. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing. 2021, 2579−2591
58. Xia S, Xiong J, Dong H, Zhao J, Tian Y, Zhou M, He Y, Han S, Zhang D. Vision language models for spreadsheet understanding: challenges and opportunities. In: Proceedings of the 3rd Workshop on Advances in Language and Vision Research (ALVR). 2024, 116−128
59. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition. 2016, 770−778
60. Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, Uszkoreit J, Houlsby N. An image is worth 16x16 words: transformers for image recognition at scale. In: Proceedings of International Conference on Learning Representations. 2021
61. Devlin J, Chang M W, Lee K, Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of 2019 Conference of the North American Chapter of the Association for Computational Linguistics. 2019, 4171−4186
62. Iida H, Thai D, Manjunatha V, Iyyer M. TABBIE: pretrained representations of tabular data. In: Proceedings of 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2021, 3446−3456
63. Li H, Zhang J, Li C, Chen H. RESDSQL: decoupling schema linking and skeleton parsing for text-to-SQL. In: Proceedings of the 37th AAAI Conference on Artificial Intelligence. 2023, 13067−13075
64. Wei Y, Wang Z, Liu J, Ding Y, Zhang L. Magicoder: empowering code generation with OSS-instruct. In: Proceedings of the 41st International Conference on Machine Learning. 2024
65. Yang J, Hui B, Yang M, Yang J, Lin J, Zhou C. Synthesizing text-to-SQL data from weak and strong LLMs. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics. 2024, 7864−7875
66. Zhang C, Mao Y, Fan Y, Mi Y, Gao Y, Chen L, Lou D, Lin J. FinSQL: model-agnostic LLMs-based text-to-SQL framework for financial analysis. In: Proceedings of Companion of 2024 International Conference on Management of Data. 2024, 93−105

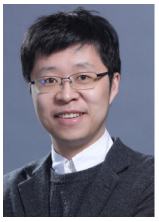
67. Zhang X, Zhang J, Ma Z, Li Y, Zhang B, Li G, Yao Z, Xu K, Zhou J, Zhang-Li D, Yu J, Zhao S, Li J, Tang J. TableLLM: enabling tabular data manipulation by LLMs in real office usage scenarios. 2024, arXiv preprint arXiv: 2403.19318
68. Zhuang A, Zhang G, Zheng T, Du X, Wang J, Ren W, Huang S W, Fu J, Yue X, Chen W. StructLM: towards building generalist models for structured knowledge grounding. 2024, arXiv preprint arXiv: 2402.16671
69. Fan J, Gu Z, Zhang S, Zhang Y, Chen Z, Cao L, Li G, Madden S, Du X, Tang N. Combining small language models and large language models for zero-shot NL2SQL. Proceedings of the VLDB Endowment, 2024, 17(11): 2750−2763
70. Alonso I, Agirre E, Lapata M. PixT3: pixel-based table-to-text generation. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics. 2024, 6721−6736
71. Parmar J, Satheesh S, Patwary M, Shoeybi M, Catanzaro B. Reuse, don't retrain: a recipe for continued pretraining of language models. 2024, arXiv preprint arXiv: 2407.07263
72. Li Z, Peng B, He P, Galley M, Gao J, Yan X. Guiding large language models via directional stimulus prompting. In: Proceedings of the 37th International Conference on Neural Information Processing System. 2023, 2735
73. Luo Z, Xu C, Zhao P, Sun Q, Geng X, Hu W, Tao C, Ma J, Lin Q, Jiang D. WizardCoder: empowering code large language models with Evol-Instruct. In: Proceedings of the 12th International Conference on Learning Representations. 2024
74. Rafailov R, Sharma A, Mitchell E, Ermon S, Manning C D, Finn C. Direct preference optimization: your language model is secretly a reward model. In: Proceedings of the 37th International Conference on Neural Information Processing System. 2023, 2338
75. Lester B, Al-Rfou R, Constant N. The power of scale for parameter-efficient prompt tuning. In: Proceedings of 2021 Conference on Empirical Methods in Natural Language Processing. 2021, 3045−3059
76. Hu E, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, Wang L, Chen W. LoRA: lowrank adaptation of large language models. In: Proceedings of the 10th International Conference on Learning Representations. 2022
77. Li B, Luo Y, Chai C, Li G, Tang N. The dawn of natural language to SQL: are we fully ready? Proceedings of the VLDB Endowment, 2024, 17(11): 3318−3331
78. Li R, Allal L B, Zi Y, Muennighoff N, Kocetkov D, Mou C, Marone M, Akiki C, Li J, Chim J, Liu Q, Zheltonozhskii E, Zhuo T Y, Wang T, Dehaene O, Davaadorj M, Lamy-Poirier J, Monteiro J, Shliazhko O, Gontier N, Meade N, Zebaze A, Yee M H, Umapathi L K, Zhu J, Lipkin B, Oblokulov M, Wang Z R, Murthy R, Stillerman J, Patel S S, Abulkhanov D, Zocca M, Dey M, Zhang Z, Fahmy N, Bhattacharyya U, Yu W, Singh S, Luccioni S, Villegas P, Kunakov M, Zhdanov F, Romero M, Lee T, Timor N, Ding J, Schlesinger C, Schoelkopf H, Ebert J, Dao T, Mishra M, Gu A, Robinson J, Anderson C J, Dolan-Gavitt B, Contractor D, Reddy S, Fried D, Bahdanau D, Jernite Y, Ferrandis C M, Hughes S, Wolf T, Guha A, von Werra L, de Vries H. StarCoder: may the source be with you! Transactions on Machine Learning Research, 2023
79. Ji Z, Lee N, Frieske R, Yu T, Su D, Xu Y, Ishii E, Bang Y J, Madotto A, Fung P. Survey of hallucination in natural language generation. ACM Computing Surveys, 2023, 55(12): 248
80. Nassar A, Livathinos N, Lysak M, Staar P. TableFormer: table structure understanding with transformers. In: Proceedings of 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022, 4604−4613
81. Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K R, Cao Y. ReAct: synergizing reasoning and acting in language models. In: Proceedings of the 11th International Conference on Learning Representations. 2023
82. Zhao Y, Chen L, Cohan A, Zhao C. TaPERA: enhancing faithfulness and interpretability in long-form table QA by content planning and execution-based reasoning. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics. 2024, 12824−12840
83. Zhang Z, Gao Y, Lou J G. E^5: zero-shot hierarchical table analysis using augmented LLMs via explain, extract, execute, exhibit and extrapolate. In: Proceedings of 2024 Conference of the North American Chapter of the Association for Computational Linguistics. 2024, 1244−1258
84. Zhou D, Schaerli N, Hou L, Wei J, Scales N, Wang X, Schuurmans D, Cui C, Bousquet O, Le Q V, Chi E H. Least-to-most prompting enables complex reasoning in large language models. In: Proceedings of the 11th International Conference on Learning Representations. 2023
85. Pourreza M, Rafiei D. DIN-SQL: decomposed in-context learning of text-to-SQL with self-correction. In: Proceedings of the 37th Conference on Neural Information Processing Systems. 2023
86. Xie Y, Jin X, Xie T, Matrixmxlin M, Chen L, Yu C, Lei C, Zhuo C, Hu B, Li Z. Decomposition for enhancing attention: improving LLM-based text-to-SQL through workflow paradigm. In: Proceedings of Findings of the Association for Computational Linguistics: ACL 2024. 2024, 10796−10816
87. Nahid M, Rafiei D. TabSQLify: enhancing reasoning capabilities of LLMs through table decomposition. In: Proceedings of 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2024, 5725−5737
88. Wang X, Wei J, Schuurmans D, Le Q V, Chi E H, Narang S, Chowdhery A, Zhou D. Self-consistency improves chain of thought reasoning in language models. In: Proceedings of the 11th International Conference on Learning Representations. 2023
89. Lee D, Park C, Kim J, Park H. MCS-SQL: leveraging multiple prompts and multiple-choice selection for text-to-SQL generation. 2024, arXiv preprint arXiv: 2405.07467
90. Jiang S, Wang Y, Wang Y. SelfEvolve: a code evolution framework via large language models. 2023, arXiv preprint arXiv: 2306.02907
91. Karpukhin V, Oguz B, Min S, Lewis P, Wu L, Edunov S, Chen D, Yih W T. Dense passage retrieval for open-domain question answering. In: Proceedings of 2020 Conference on Empirical Methods in Natural Language Processing. 2020, 6769−6781
92. Jiang J, Zhou K, Dong Z, Ye K, Zhao X, Wen J R. StructGPT: a general framework for large language model to reason over structured data. In: Proceedings of 2023 Conference on Empirical Methods in Natural Language Processing. 2023, 9237−9251
93. Sui Y, Zou J, Zhou M, He X, Du L, Han S, Zhang D M. TAP4LLM: table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. 2023, arXiv preprint arXiv: 2312.09039
94. Chen S, Liu H, Jin W, Sun X, Feng X, Fan J, Du X, Tang N. ChatPipe: orchestrating data preparation pipelines by optimizing human-ChatGPT interactions. In: Proceedings of Companion of 2024 International Conference on Management of Data. 2024, 484−487
95. Fan J, Wang Z, Xie Y, Yang Z. A theoretical analysis of deep Q-learning. In: Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control. 2020, 486−489
96. Zhao B, Ji C, Zhang Y, He W, Wang Y, Wang Q, Feng R, Zhang X. Large language models are complex table parsers. In: Proceedings of 2023 Conference on Empirical Methods in Natural Language Processing. 2023, 14786−14802
97. Li J, Huo N, Gao Y, Shi J, Zhao Y, Qu G, Wu Y, Ma C, Lou J G, Cheng R. Tapilot-crossing: benchmarking and evolving LLMs towards interactive data analysis agents. 2024, arXiv preprint arXiv: 2403.05307v1

98. Zhong V, Xiong C, Socher R. Seq2SQL: generating structured queries from natural language using reinforcement learning. 2017, arXiv preprint arXiv: 1709.00103
99. Zhao Y, Zhao C, Nan L, Qi Z, Zhang W, Tang X, Mi B, Radev D. RobuT: a systematic study of table QA robustness against human-annotated adversarial perturbations. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics. 2023, 6064−6081
100. Iyyer M, Yih W T, Chang M W. Search-based neural structured learning for sequential question answering. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. 2017, 1821−1831
101. Li J, Hui B, Qu G, Yang J, Li B, Li B, Wang B, Qin B, Geng R, Huo N, Zhou X, Ma C, Li G, Chang K C C, Huang F, Cheng R, Li Y. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. 2023, 1835
102. Motl J, Schulte O. The CTU Prague relational learning repository. 2024, arXiv preprint arXiv: 1511.03086
103. Chang S, Wang J, Dong M, Pan L, Zhu H, Li A, Lan W, Zhang S,·Jiang J, Lilien J, Ash S, Wang W,·Wang Z,·Castelli V, Ng P,·Xiang B. Dr.Spider: a diagnostic evaluation benchmark towards text-to-SQL robustness. In: Proceedings of the 11th International Conference on Learning Representations. 2023
104. Zhang Y, Deriu J, Katsogiannis-Meimarakis G, Kosten C, Koutrika G, Stockinger K. ScienceBenchmark: a complex real-world benchmark for evaluating natural language to SQL systems. Proceedings of the VLDB Endowment, 2024, 17(4): 685−698
105. He X, Zhou M, Zhou M, Xu J, Lv X, Li T, Shao Y, Han S, Yuan Z, Zhang D. AnaMeta: a table understanding dataset of field metadata knowledge shared by multi-dimensional data analysis tasks. In: Proceedings of Findings of the Association for Computational Linguistics: ACL 2023. 2023, 9471−9492
106. Jiménez-Ruiz E, Hassanzadeh O, Efthymiou V, Chen J, Srinivas K. SemTab 2019: resources to benchmark tabular data to knowledge graph matching systems. In: Proceedings of the 17th International Conference on the Semantic Web. 2020, 514−530
107. Hulsebos M, Demiralp Ç, Groth P. GitTables: a large-scale corpus of relational tables. Proceedings of the ACM on Management of Data, 2023, 1(1): 30
108. Döhmen T, Geacu R, Hulsebos M, Schelter S. SchemaPile: a large collection of relational database schemas. Proceedings of the ACM on Management of Data, 2024, 2(3): 172
109. Wretblad N, Riseby F, Biswas R, Ahmadi A, Holmström O. Understanding the effects of noise in text-to-SQL: an examination of the BIRD-bench benchmark. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics. 2024, 356−369
110. Liu J. LlamaIndex. See Docs.llamaindex.ai/en/stable/ website, 2022
111. Xue S, Qi D, Jiang C, Cheng F, Chen K, Zhang Z, Zhang H, Wei G, Zhao W, Zhou F, Yi H, Liu S, Yang H, Chen F. Demonstration of DB-GPT: next generation data interaction system empowered by large language models. Proceedings of the VLDB Endowment, 2024, 17(12): 4365−4368
112. Vanna. AI. Vanna. See Github.com/vanna-ai/vanna website, 2023
113. Venturi G. Pandas-ai. See Github.com/Sinaptik-AI/pandas-ai website, 2023
114. Pang C, Cao Y, Yang C, Luo P. Uncovering limitations of large language models in information seeking from tables. In: Proceedings of Findings of the Association for Computational Linguistics: ACL 2024. 2024, 1388−1409
115. Touvron H, Lavril T, Izacard G, Martinet X, Lachaux M A, Lacroix T, Rozière B, Goyal N, Hambro E, Azhar F, Rodriguez A, Joulin A, Grave E, Lample G. LLaMA: open and efficient foundation language models. 2023, arXiv preprint arXiv: 2302.13971
116. Kwon W, Li Z, Zhuang S, Sheng Y, Zheng L, Yu C H, Gonzalez J, Zhang H, Stoica I. Efficient memory management for large language model serving with PagedAttention. In: Proceedings of the 29th Symposium on Operating Systems Principles. 2023, 611−626
117. Kahneman D. Thinking, Fast and Slow. London: Farrar, Straus and Giroux, 2011

Weizheng Lu is a senior research engineer at Renmin University of China. His current research interests include high-performance data science.



Jing Zhang is a professor at School of Information, Renmin University of China. Her research focuses on data mining and knowledge discovery.



Ju Fan is a professor at School of Information, Renmin University of China. His research focuses on artificial intelligence for databases.

Zihao Fu is a senior AI product manager at Kingsoft Office, specializing in spreadsheet AI. He focuses on AI-powered productivity tools and software.

Yueguo Chen is a professor at School of Information, Renmin University of China. He focuses on the interdisciplinary fields of big data and artificial intelligence with social science.

Xiaoyong Du is a professor at School of Information, Renmin University of China. His current research interests include databases and intelligent information retrieval.