SheetAgent: Towards a Generalist Agent for Spreadsheet Reasoning and Manipulation via Large Language Models

Anonymous Author(s)

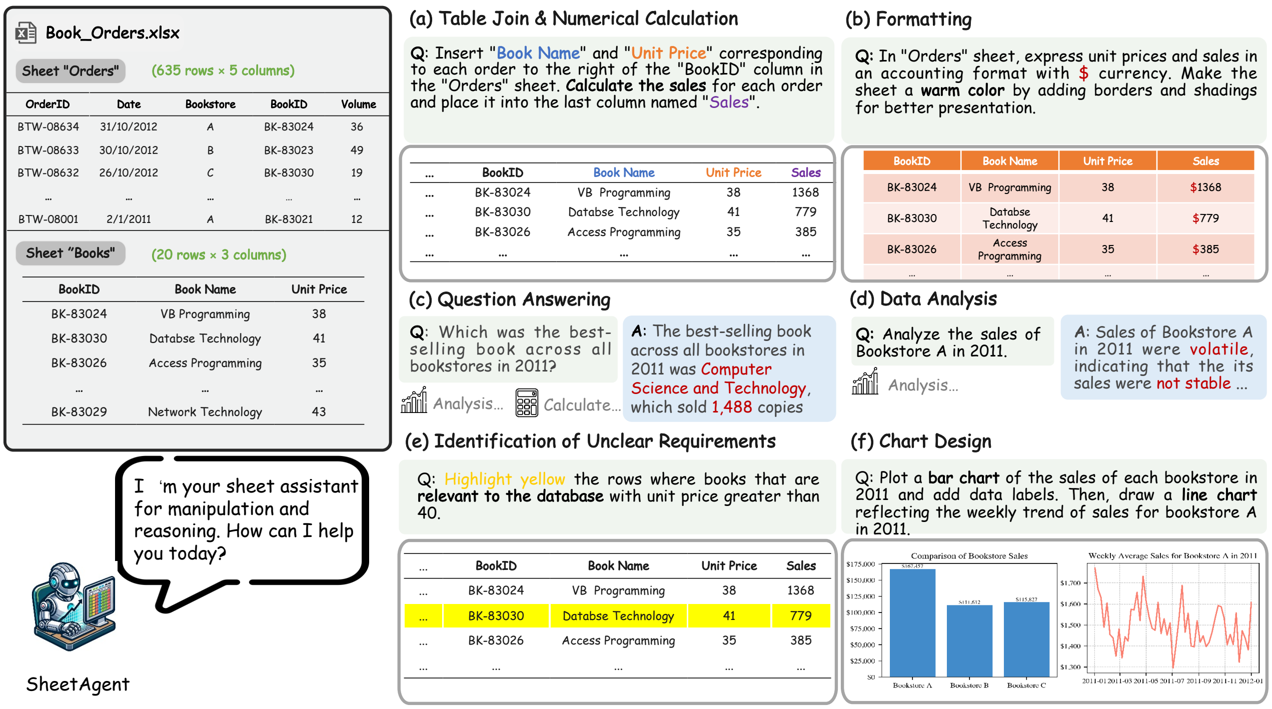


Figure 1: SheetAgent can handle diverse spreadsheet reasoning and manipulation tasks automatically. Given a large-scale spreadsheet with multiple sheets, SheetAgent showcases its proficiency in visualization (f), achieves accurate manipulation on long horizon and multi-step tasks (a, b) with consistent reasoning capabilities (c, d), even faced with the challenges like unclear requirements (e).

Abstract

Spreadsheets are ubiquitous across the World Wide Web, playing a critical role in enhancing work efficiency across various domains. Large language model (LLM) has been recently attempted for automatic spreadsheet manipulation but has not yet been investigated in complicated and realistic tasks where reasoning challenges exist (e.g., long horizon manipulation with multi-step reasoning and ambiguous requirements). To bridge the gap with the real-world requirements, we introduce **SheetRM**, a benchmark featuring long-horizon and multi-category tasks with reasoning-dependent manipulation caused by real-life challenges. To mitigate the above challenges, we further propose **SheetAgent**, a novel autonomous agent that utilizes the power of LLMs. SheetAgent consists of three collaborative modules: *Planner*, *Informer*, and *Retriever*, achieving both advanced reasoning and accurate manipulation over spreadsheets without human interaction through iterative task reasoning and reflection. Extensive experiments demonstrate that SheetAgent delivers 20–40% pass rate improvements on multiple benchmarks over baselines, achieving enhanced precision in spreadsheet manipulation and demonstrating superior table reasoning abilities. More details and visualizations are available at the project website. The datasets and source code are available at https://anonymous.4open.science/r/SheetAgent.

WWW '25, April 28 – May 2, 2025, Sydney, Australia

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

CCS Concepts

• **Computing methodologies → Natural language processing**; • **Information systems → Decision support systems**.

Keywords

Agents, Large Language Models, Benchmark, Spreadsheet Reasoning and Manipulation

1 Introduction

Tabular data plays a crucial role in domains such as scientific research, finance, and marketing, where it is predominantly handled using spreadsheet systems. These systems, such as Google Sheets and Microsoft Excel, are useful for tasks including numerical calculations, data analysis, and visualisation [10, 14, 19]. However, processing these affairs often involves a significant amount of repetitive labor and consultation [7, 11]. Recent work [7, 20] has explored the automation of simple spreadsheet manipulation tasks. For example, *Highlight rows with sales volume greater than 40* in the "Orders" sheet shown in Figure 1. This task can be accomplished through simple queries and formatting. However, they fail to consider the more complex and realistic tasks that encompass more than simple first-order logic. For instance, *Highlight rows of database-related books with sales volume greater than 40*. The difficulty of this instruction lies in identifying books related to the database, which cannot be achieved simply by understanding the semantics of column names, but rather by perceiving the specific content (what books are in the spreadsheet in this case) of the spreadsheet. Such scenarios are common because a complete spreadsheet task may demand a multi-step reasoning process in conjunction with multiple sheets, and the user may not precisely define the required operations or ambiguously interpret the task instruction. Consequently, there is an urgent need for a new method to automate these tasks.

Designing such a method demands a combination of sophisticated sheet-based reasoning and manipulation capabilities. Previous work [13, 20, 26] has focused on precise spreadsheet manipulation while neglecting reasoning, limiting them to tasks with clear expressions and one-step reasoning. The emergence of large language models (LLMs) like GPTs [2, 28, 29] enables the integration of reasoning and manipulation capabilities. Extensive research [5, 17, 39] has shown that LLMs can reason over tables, handling tasks such as table question answering and fact verification. Given this context, we are motivated to explore the question: *Can we build a versatile agent adept at handling complex spreadsheet manipulation tasks with challenging reasoning factors using LLMs?* Crafting such an agent involves several challenges: (1) **Dynamic Changes of Spreadsheet**: Complex tasks often involve multiple operations, resulting in dynamic spreadsheet content changes. Continuously feeding the entire spreadsheet into LLMs before each operation is impractical due to token limits and potential hallucination [8, 39]. (2) **Limited Table Understanding**: LLMs are predominantly trained in natural languages and show limited understanding of tables [21]. (3) **Lack of Benchmark**: There is an absence of a complicated benchmark demanding accurate reasoning and precise manipulation over spreadsheets. SheetCopilot [20] presents a benchmark for evaluating LLM performance in controlling spreadsheets. However, it simplifies real-world requirements, ignoring challenges like multi-step reasoning and long-horizon operations.

To address the dataset gap, we first introduce **SheetRM** (**Spreadsheet R**easoning and **M**anipulation Benchmark), a benchmark for developing and evaluating LLM-based agents for precise spreadsheet manipulation and advanced reasoning capabilities. Each task in SheetRM involves multiple subtasks that relies on reasoning abilities, derived from real-world Excel exam datasets. Moreover, it enables automatic evaluation with various metrics. We further present **SheetAgent**, a generalist agent for sheet manipulation and reasoning using LLMs. SheetAgent mainly consists of three components: the Planner, Informer, and Retriever. The Planner translates conceptual understandings into proficient code generation to manipulate spreadsheets. The Informer parses task demands and produces high-quality, task-specific SQL queries to understand the spreadsheet without needing to read the entire table, despite its dynamic changes. The Retriever retrieves instructive examples to improve the robustness of solutions. We demonstrate that SheetAgent significantly outperforms other state-of-the-art baselines in diverse benchmarks. Our contributions are three-fold:

- We introduce SheetRM, a benchmark for developing and evaluating LLM-based agents to manipulate spreadsheets with advanced reasoning abilities. It includes more challenging tasks that reflect real-world requests and supports automatic evaluation with various metrics.
- We develop a versatile LLM-based agent SheetAgent, combining sheet manipulation and reasoning abilities to boost multifaceted interaction between humans and spreadsheets.
- Experimental results show that SheetAgent can be combined with any LLMs backbone and SheetAgent outperforms baselines across multiple benchmarks, achieving a 20–40% improvement in various metrics. These results highlight SheetAgent's exceptional capabilities in spreadsheet manipulation and table reasoning.

2 SheetRM Benchmark

Unlike existing datasets [12, 20, 26] primarily designed for more precise spreadsheet manipulation, our goal is to construct a more realistic dataset, where tasks contain challenges such as complicated multi-step reasoning and vague requirements, to narrow the gap between simulation and real-world scenarios. To achieve this, we begin by sourcing spreadsheets from real-life Excel exam datasets. We collate a diverse set of spreadsheet operations commonly used in realistic scenarios and analyze the challenges faced when addressing spreadsheet tasks in practical settings. In brief, our SheetRM dataset is featured by the following elements, as outlined in Figure 2(a)-(d):

- **Multi category:** We summarize and collect 5 broad types and 36 subtypes of manipulation categories with corresponding 4 reasoning challenges. Each task includes an examination of both manipulation and reasoning abilities.
- **Reasoning-dependent manipulation:** Tasks include operations with multi-step reasoning over spreadsheets.
- **Long horizon:** Various subtasks constitute a complete task, which brings to agents the challenge of dynamic changes in spreadsheets.
- **Procedure evaluation:** We build an automated program evaluation approach for SheetRM that not only automates the determination of whether the full task is completed but also detects the completion of individual subtasks.

2.1 Task Schema

Each task is defined by the following three parts. See Figure 2 (upper right) for a visual demonstration.

Spreadsheet Assets. Each task consists of a spreadsheet as well as multiple sheets. We summarize the contents of the spreadsheet

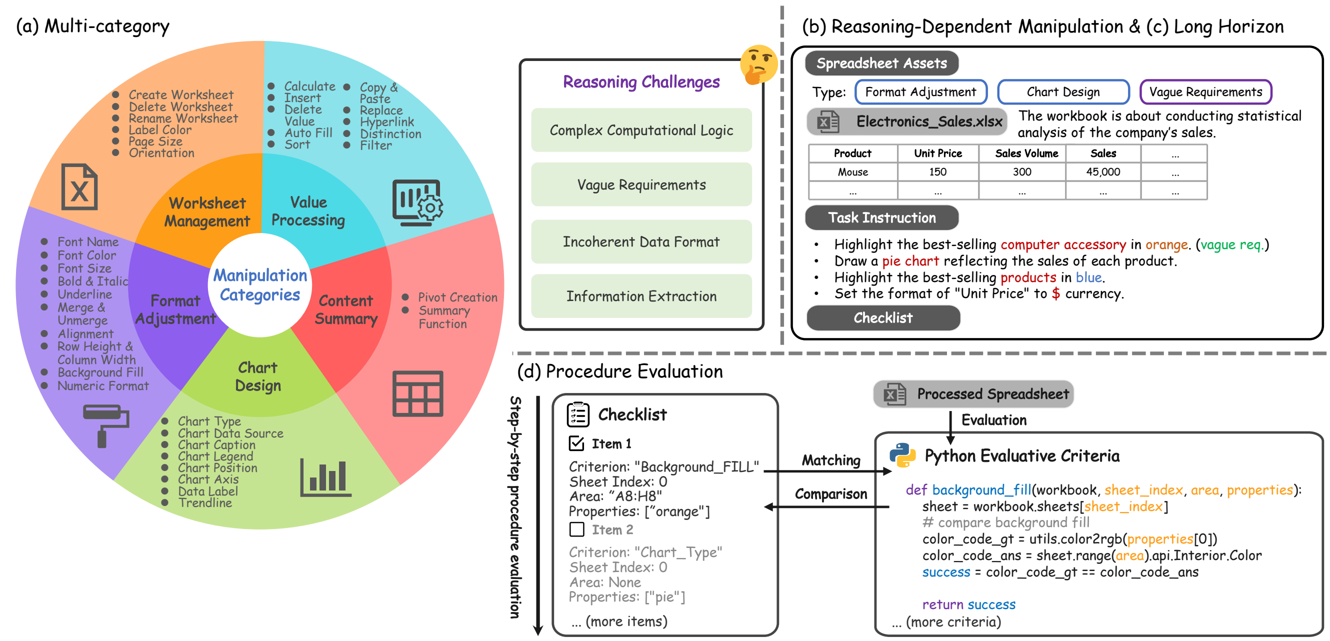


Figure 2: Overview and features of SheetRM. (a) Multi-category: SheetRM contain real-life tasks for multiple types of manipulation categories and reasoning challenges. Each task includes an examination of both manipulation and reasoning abilities. (b&c) Long horizon and reasoning-dependent Manipulation: An example task including three parts. Spreadsheet assets contain sheet data and one-sentence description with category of tasks. Then task instruction provides the requirements for the execution of the long horizon tasks. Checklist is designed for procedure evaluation. (d) Procedure evaluation: SheetRM automatically evaluates each task step-by-step via corresponding checklist and evaluative criterion to achieve procedure evaluation.

in a one-sentence natural language overview as context, as shown in Table 7, aiming to stimulate the internal knowledge of LLMs.

Task Instruction. A task instruction outlines the overall requirements expressed in natural language. Completing a task instruction requires a series of operations on the target spreadsheet.

Checklist. A task is paired with a checklist designed to evaluate its completion. Each item in the checklist corresponds to the evaluation of a fine-grained operation with tailored criteria. The automatic evaluation will be discussed in detail in Section 2.3.

2.2 Dataset Construction

We gather and refine publicly available spreadsheets through a selection and cleaning process. Tasks are generated with both human and GPT-4 annotation. All the tasks are attached with verified answers, which enables model-free evaluation. The statistics of our curated dataset are shown in Table 1. Compared to the SheetCopilot benchmark, our SheetRM has a more granular and reasonable categorisation, holds more tasks with longer horizon and includes reasoning challenges. See Appendix B for a detailed comparison and more statistics.

Spreadsheet Files Collection. We initially collect real-world spreadsheets[1] from a public examination question bank, filtering out files that are protected, corrupted, or inaccessible. To minimize privacy risks, we further modify sensitive information, such as adding noise to age data and anonymizing bookstore names. We select files covering multiple domains to ensure diversity, ensuring most files have at least 2 sheets, with a minimum of 20 rows and 5 columns. External dependencies are converted into natural language or embedded sheets if feasible. We finally shortlist 41 spreadsheets with a total of 137 sheets. On average, each spreadsheet contains 300.82 rows and 26.23 columns. For more collection details, we refer to Appendix B.1.

Task Generation. We begin by referring to websites about spreadsheet software skills and consult corporate staff about commonly used spreadsheet operations in their work. As shown in Figure 2, we conclude five coarse operation categories and their fine-grained specific operations for manipulation. Drawing insights from common table reasoning datasets like WikiTableQuestions, FeTaQA, and TabFact, we summarize four challenges in the process of sheet reasoning: (1) complex computation logic, (2) vague requirements,

Table 1: Basic statistics of SheetRM.

Item	Count
# Sheets	137
# Average Rows per File	300.82
# Average Columns per File	26.23
# Tasks	317
# Subtasks	1625

[1]A spreadsheet is a collection of sheets that are organized into a document. A table represents a structured arrangement of data in rows and columns. Each sheet within the spreadsheet contains a table.

(3) incoherent data format and (4) information extraction. We detail these challenges in the Appendix B.4. Then, we instruct GPT-4 to propose *realistic* tasks that mimic user requests adhering to four guidelines: the tasks should only involve predefined operations, cover diverse manipulation categories, exhibit a long-horizon nature by encompassing multiple subtasks, and incorporate at least one subtask that presents the specified reasoning challenges. This process yields a compilation of 2316 subtasks. We eliminate semantically redundant entries for identical files to maintain uniqueness. To guarantee quality, our internal annotators manually validate subtasks using programming and specialized software such as Excel. Certain unreasonable subtasks are excluded throughout this process. By combining these subtasks for different spreadsheets considering horizon and complexity, we ultimately assemble 317 task instructions, encompassing a total of 1625 subtasks. Full prompts are available in Appendix I.1.

2.3 Automatic Evaluation

SheetCopilot [20] introduces a feasible method that determines task fulfillment by evaluating the alignment of key properties between the processed spreadsheet and ground truth candidates. However, this method fails to evaluate the accuracy of each operation involved, as a task may comprise multiple detailed intermediate steps. To address this challenge, we develop an automatic evaluation system that is model-free and tailored for each fine-grained operation. The advantage of that is that we are able to evaluate the performance of intermediate sub-task processes. A checklist is crafted for each task instruction. As illustrated in Figure 2, within the checklist, each evaluation item comprises a `(Criterion, Sheet Index, Area, Properties)` pair. We locate the comparison region in the target spreadsheet by `(Sheet Index, Area)`. Then, corresponding `Criterion` is applied to verify whether the region aligns with the `Properties`. This design enables a detailed evaluation of LLMs' capabilities by assessing each step of task execution.

3 SheetAgent Framework

To quantify the challenges posed by SheetRM, we introduce an LLM-based agent framework SheetAgent. As outlined in Figure 3, SheetAgent consists of three key components: the Planner, the Informer, and the Retriever. The Planner generates Python code to manipulate the target spreadsheet, reasoning and acting to address complex tasks. The Informer supplies task-specific SQL queries, whose execution results provide crucial subviews of the spreadsheet, narrowing the reasoning scope for the Planner and enhancing its ability to tackle complicated spreadsheet reasoning challenges. At each decision step, the Planner can perform more precise manipulations with subviews. When the Planner generates an incorrect solution, the Retriever is activated to fetch high-quality code examples from our curated repository, assisting the Planner in making more effective corrections. Pseudo code is provided in Appendix A.

3.1 Proficient Spreadsheet Manipulation with Planner

We design a Planner module to manipulate spreadsheets in SheetAgent. The way to interact with spreadsheets determines the precision of manipulation. Unlike SheetCopilot [20], which uses a set of language APIs, we adopt a code-centric approach to control spreadsheets. We find Python, compared with VBA, is more suitable for manipulating spreadsheets due to its alignment with the training corpora of most existing LLMs [4, 30]. This Python code-centric approach reduces the occurrence of hallucinations of LLMs. We refer to Appendix C for details on the code-centric design.

Complex spreadsheet manipulation tasks often involve multiple steps. Achieving precise control over spreadsheets is challenging without an effective feedback mechanism. To address this, we devise a closed-loop planning process where the Planner interacts with the target spreadsheet, incorporating feedback and reflection. We first concatenate task instruction I, system prompt P^P, description D, and the initial sheet state s_0 (row and column count, headers, and data type of each column) as the input. Given a snapshot of the target spreadsheet at step t, the Planner generates action $a_t = \text{Planner}(a_t|I, P^P, D, s_t, h_{t-1})$, where h_{t-1} is the planning history. The action is evaluated in a sandbox with the feedback $o_t = \text{Sandbox}(a_t)$. If an error occurs, the Planner reflects and generates an adjusted action $a_t^* = \text{Planner}(a_t^*|I, P^P, D, s_t, h_{t-1}, o_t)$. Otherwise, the action is performed on the target spreadsheet. The spreadsheet is updated to a new state of s_{t+1}. The planning history is also updated to $h_t = (h_{t-1}, o_t, a_t)$. By this, the Planner can achieve accurate manipulation with only the key information (i.e., the sheet state) of the target spreadsheet rather than reading all the sheet data.

3.2 Accurate Spreadsheet Perception with Informer

Merely being aware of the sheet state does not equip the Planner to address the reasoning challenges shown in Figure 2. For instance, to fulfill the instruction illustrated in Figure 3, the Planner needs to discern which products qualify as computer accessories. However, the Planner struggles to query the spreadsheet effectively due to the absence of efficient mechanisms like SQLs and lacks the intrinsic ability to comprehend the data's semantics. A feasible approach is constantly feeding the full spreadsheet into the Planner. However, considering the continuity of operations in a complicated task, a spreadsheet may experience multiple modifications, making it challenging to maintain a synchronized state within the Planner due to the token limit.

Therefore, we introduce the Informer to handle table content of arbitrary length and dynamic changes. Informer generates task-specific SQLs to perform queries. Initially, the tabular data in the target spreadsheet is extracted and stored in a lightweight database. At step t, the Informer's objective is to select entries that align best with both the task instruction and the current step. To achieve this, we formulate the input of the Informer by combining the system prompt P^I, the task instruction I, and previous actions $A_{t-1} = (a_1, \cdots, a_{t-1})$ performed by the Planner. Then, the Informer generates an SQL query $q_t = \text{Informer}(q_t|I, P^I, A_{t-1}, s_t)$. A_{t-1} functions as a reasoning trace of the Planner, enabling the Informer to generate more task-specific and robust SQLs. As shown in Figure 3, the execution result of the SQL query is a spreadsheet subview, which serves as evidence e_t for the Planner to reason over. This allows the Planner to more accurately and efficiently perceive the target spreadsheet from key evidence, thereby addressing the

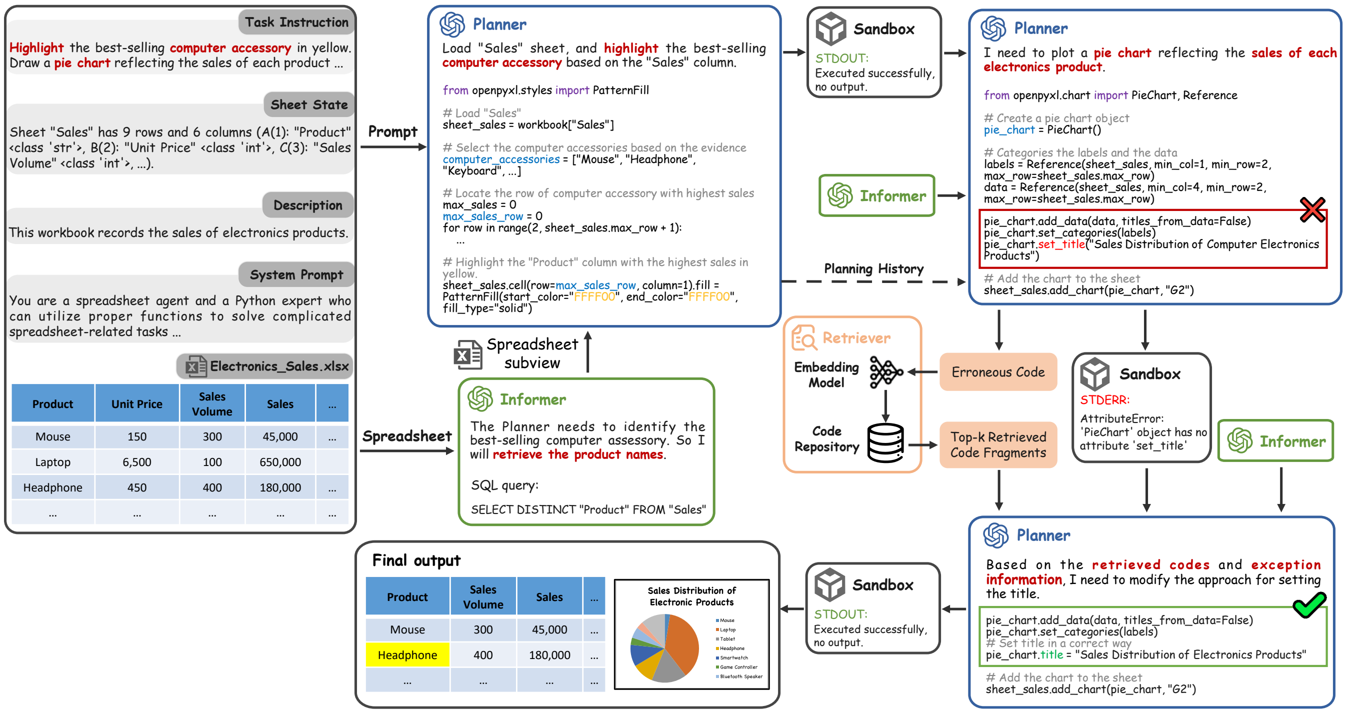


Figure 3: An illustration of SheetAgent. SheetAgent comprises three key components, including the Planner, the Informer, and the Retriever. The Planner interacts with the target spreadsheet via a virtual sandbox. The Informer provides subtask-specific SQLs, the execution results of which serve as the evidence for the Planner to handle reasoning challenges. The Retriever is invoked to retrieve similar tutorial code snippets upon encountering an error, effectively correcting the error.

reasoning challenges. Note that each time the Planner performs an operation, the spreadsheet in the database is updated to maintain synchronization.

3.3 Robust Solution Generation with Retriever

The Retriever advises the Planner during task planning, augmenting error corrections by sourcing relevant code from a code repository. We collect high-quality code from GitHub and craft tutorial examples for each manipulation category shown in Figure 2. We organize them into a compilation of code files. To improve efficiency, we employ Milvus [34], an open-source vector database, as the code repository. To construct this repository, a sliding window technique is applied to traverse these files, extracting continuous lines of code within the window size. These code fragments $\mathcal{C}_{repo}$ are embedded into a set of vectors and stored in the code repository. The Retriever is invoked when the sandbox emits an error signal. We seek top-k similar code snippets $\mathcal{C}^k_{ret}$ as follows:

$$\mathcal{C}^k_{ret} = \Big\{ C^i_{repo} | C^i_{repo} \in \mathcal{C}_{repo}, \forall C^j_{repo} \notin \mathcal{C}^k_{ret}, \\ \text{sim}\left(\mathcal{E}(C_q), \mathcal{E}(C^i_{repo})\right) > \text{sim}\left(\mathcal{E}(C_q), \mathcal{E}(C^j_{repo})\right) \Big\},$$

wherein $|\mathcal{C}^k_{ret}| = k$, C_q refers to the erroneous code snippet, and $\text{sim}(\cdot)$ denotes cosine similarity. The embedding function[2] $\mathcal{E}(\cdot)$ can be represented by any pretrained language model. Consequently, the top-k similar code snippets $\mathcal{C}^k_{ret}$ arranged in descending order are retrieved. These code snippets boost the replanning process of the Planner with $a^*_t = \text{Planner}(a^*_t | I, P^P, D, s_t, h_{t-1}, o_t, \mathcal{C}^k_{ret})$ for generating more robust and reliable solutions. We provide details of code collection in Appendix D.

[2]Here we use text-embedding-ada-002. See https://platform.openai.com/docs/models/embeddings.

4 Experiment

We conduct experiments on various tasks to answer the following research questions (RQs):
Versatility (RQ1): Is SheetAgent adept at both spreadsheet manipulation and reasoning?
Universality (RQ2): Can different LLMs benefit from the design of SheetAgent?
Difficulty (RQ3): Why SheetRM is a challenging benchmark for existing methods?
Ablation (RQ4): How do the modules within SheetAgent contribute to its overall effectiveness?

4.1 Experiment Setup

Dataset and Evaluation Metrics. We evaluate our approach SheetAgent on 5 diverse benchmarks. *SheetCopilot Benchmark* (SCB), a benchmark consisting of 221 tasks, is selected to mainly assess the manipulation ability. To measure the reasoning capability, we adopt three table reasoning tasks, including *WikiTableQuestions* (WTQ) [25], *FeTaQA* [23], and *TabFact* [6]. We report the performance on these tasks using their official evaluation pipeline. The 317 tasks

Table 2: Performance comparison of different methods on SCB and our SheetRM. * denotes results on a subset of SCB with 20 representative tasks. Best results are bolded and second-best results are underlined.

Method	SCB		SheetRM		
	Exec@1↑	Pass@1↑	Exec@1↑	Pass@1↑	SubPass@1↑
VBA (GPT-3.5)	77.8	37.1	56.2	2.8	14.5
OS-Copilot (GPT-4)	/	60.0*	74.8	20.5	56.6
SheetCopilot (GPT-4)	65.0*	55.0*	68.1	0	16.2
SheetCopilot (GPT-3.5)	87.3	44.3	52.7	2.2	30.7
SheetAgent (GPT-3.5)	**94.1**	**61.1**	89.3	**44.8**	**77.0**
- *w/o* Informer+Retriever	88.7	50.7	88.6	11.4	57.5
SheetAgent (GPT-4)	**90.0***	**70.0***	**92.4**	31.2	69.8

Table 3: Results of different methods on three table reasoning benchmarks. Best results are bolded and second-best results are underlined.

Method	WTQ	TabFact	FeTaQA
Fine-tuning based LLMs			
TAPEX [22]	57.5	84.2	34.7
UnifiedSKG [38]	49.3	**85.4**	33.4
OmniTab [18]	62.8	82.8	34.9
Prompting based LLMs			
GPT-3 CoT [5]	45.7	76.0	27.0
Binder [8]	59.9	82.9	31.6
DATER [39]	61.6	80.7	30.9
StructGPT [17]	52.2	81.2	32.5
SheetAgent (GPT-3.5)	**64.4**	84.8	**36.7**

in our *SheetRM* is used to comprehensively evaluate manipulation and reasoning capabilities. Refer to Appendix E for more details of these datasets. For manipulation tasks, we adopt Exec@1 and Pass@1 following SheetCopilot. Exec@1 measures the percentage of solutions without exceptions during execution. Pass@1 is used to evaluate the successful accomplishment of the task. In addition, we use the SubPass@1 to count the success rate of subtasks in each task to assess the instruction following capability. As for reasoning tasks, we chose distinct evaluation metrics. For WTQ and TabFact, accuracy is adopted as the evaluation metric. For FeTaQA, we report the sacreBLEU score [27].

Baselines. For SCB, we compare SheetAgent with SheetCopilot [20] and OS-Copilot [37], two LLM-based agent frameworks, and VBA [20], a method that generates and runs VBA code. For table-based reasoning tasks, we select fine-tuning based LLMs like TAPEX [22] and OmniTab [18], and prompting-based LLMs such as DATER [39] and StructGPT [17]. To our knowledge, there is a lack of methods capable of comprehensive spreadsheet manipulation. Besides, various approaches [9, 12, 31] finetune LLMs for specific tasks like formatting and formula prediction but lack open source weights. Therefore, we mainly compare VBA, SheetCopilot, and OS-Copilot on SheetRM. We chose JSON as the table representation for its superior performance as shown in Table 5. See Appendix F for implementation details.

4.2 Versatility (RQ1)

To answer RQ1, we conduct various experiments on both spreadsheet manipulation and reasoning tasks. Table 2 shows the results for SCB. Using GPT-3.5 as the backbone, we observe that SheetAgent outperforms SheetCopilot with a remarkable **16.8** higher Pass@1. Even without the Informer and Retriever components, our method still surpasses others in both metrics. This indicates that the generated Python code is more robust and reliable compared to VBA code or language APIs. Following Li et al. [20], we use GPT-4 on a subset of SCB, including 20 tasks. Our SheetAgent also outperforms SheetCopilot and OS-Copilot with **15.0** and **10.0** higher Pass@1 respectively. These results demonstrate that SheetAgent can better leverage the power of LLMs to achieve more accurate spreadsheet manipulation.

Further experiments focus on assessing SheetAgent's reasoning capability. We remove the Retriever as these tasks typically involve simpler operations like sorting and filtering. Results in Table 3 show that SheetAgent outperforms other baselines on WTQ and FeTaQA tasks, indicating its capability to provide precise and informative responses. Besides, SheetAgent surpasses all fine-tuning based methods, and achieves comparable performance as SOTA method UnifiedSKG [38] on TabFact. The results underscores the synergy between the Planner and Informer, which significantly enhances SheetAgent's efficacy in table reasoning tasks. Full results are provided in Appendix G.1.

We compare SheetAgent with baselines on the SheetRM to evaluate reasoning and manipulation capabilities. For fair comparison, we have improved SheetCopilot based on the open-source version with error feedback functionality. Results in Table 2 show SheetAgent significantly outperforms other baselines in three aspects: (1) **Robust solution generation:** SheetAgent achieves an Exec@1 of **92.4**, indicating more robust solutions, meaning Python code generated by LLMs is more robust than VBA and language APIs. (2) **Strong manipulation proficiency:** SheetAgent is proficient in complex multi-category tasks, achieving a maximum SubPass@1 of **77.0**, more than double that of SheetCopilot with GPT-4. (3) **Advanced reasoning ability:** SheetAgent can solve more reasoning challenges, whereas SheetCopilot struggles significantly (Pass@1 **44.8 vs 2.2**). This reflects the superior reasoning capabilities of SheetAgent. We also provide an illustrative case in Figure 7 to further demonstrate why SheetAgent outperforms SheetCopilot in tasks with reasoning challenges.

4.3 Universality (RQ2)

To answer RQ2, we compare our SheetAgent with SheetCopilot across various LLM backbones on SheetRM. As presented in Table 2 and Figure 4, SheetAgent shows remarkable improvements in all evaluated metrics on diverse backbones such as GPTs and Claude. Despite with smaller, open-source backbones, we can continue to observe the same results. Furthermore, SheetCopilot fails to pass any task completely on open-source models, and possesses lower Exec@1 scores, highlighting its challenges in generating feasible solutions. These results confirm the universality of SheetAgent, illustrating its consistent performance improvements across different LLM backbones regardless of scale. Meanwhile, we note that the differences between various LLM backbones mainly stem from their fundamental capabilities, such as instruction following and code

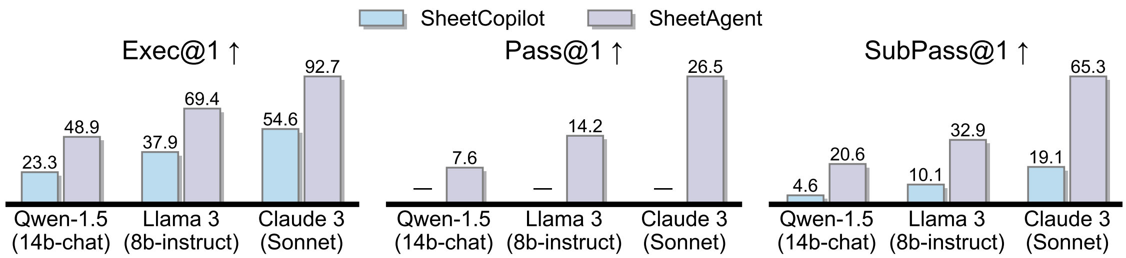
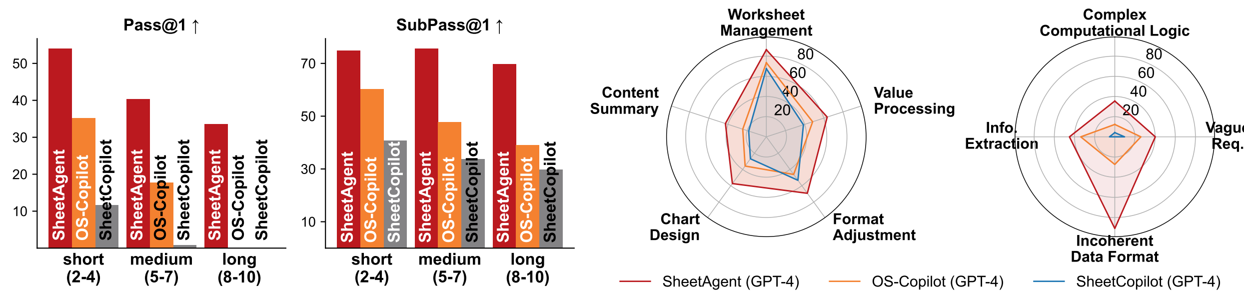


Figure 4: Performance on SheetRM for other LLM backbones. "—" means Pass@1=0. These backbones benefit significantly from the design of SheetAgent compared to SheetCopilot.



(a) **(b)**

Figure 5: Comparison between SheetAgent and SheetCopilot with GPT-4. (a) Comparison of Pass@1 and SubPass@1 under different task horizon levels. (b) Pass rate of different manipulation categories (left) and reasoning challenges (right).

generation abilities. We further analyze failure cases on SheetRM in Section 4.6 to identify the deficiencies and opportunities for improvement in SheetAgent under different LLM backbones.

4.4 Difficulty (RQ3)

We explore the challenges of our proposed SheetRM benchmark from three perspectives: task horizon, task categories, and reasoning challenges. We compare SheetAgent with SheetCopilot and OS-Copilot against the same GPT-4 backbone. Tasks are categorized into three levels based on their horizon: short (2-4), medium (5-7), and long (8-10). As depicted in Figure 5(a), both methods exhibit a decreasing trend in Pass@1 and SubPass@1 as task horizon increases, indicating the difficulty of long-horizon tasks in our benchmark. Furthermore, Figure 5(b) presents the performance across different manipulation categories and reasoning challenges by evaluating subtask success rates. Both methods struggle with more complex tasks like chart design and content summary. Additionally, SheetCopilot can hardly address reasoning challenges. These findings underscore the challenges introduced by SheetRM, particularly in domains requiring consistent and robust reasoning and manipulation capabilities. Noted that our SheetAgent still outperforms SheetCopilot, further validating its superior abilities.

4.5 Ablation (RQ4)

Effects of Each Module. Table 4 reveals the effects of SheetAgent modules. Pass@1 drops dramatically without Informer, indicating its vital role in handling reasoning challenges by providing relevant information. Exec@1 also decreases sharply without Retriever, showing that high-quality examples help the Planner generate reliable code. Without both Informer and Retriever, SheetAgent performs poorest, highlighting the need for both reasoning and manipulation capabilities to tackle complex tasks effectively. Combining the results in Table 2, even with only the Planner, SheetAgent performs decently compared to SheetCopilot, showcasing the benefits of a code-centric approach.

Table Representations. Tabular data requires reliable representations for LLMs to recognize attribute relationships. We ablate four table representations—JSON, DFLoader, Markdown, and HTML—for SheetAgent on WTQ and SheetRM. Results in Table 5 show JSON outperforms other formats. HTML performs poorly on SheetRM due to verbosity and token limits. We provide illustration of different representations in Appendix G.3 with in-depth analysis. More additional ablations about temperature and vision-enabled SheetAgent can be found in Appendix G.

4.6 Case Study

Failure Cases Analysis. In our analysis, we classified the failures of various LLMs into five categories, namely "improper function calls", "inaccurate queries", "retrieval of irrelevant code snippets", "hallucinations", and "failure to follow instructions". Figure 6 shows the distribution of these failure types across different LLM backbones. From this, we observe that smaller, open-source models (e.g., llama3-8b-instruct and qwen-14b-chat) tend to exhibit higher rates of "hallucinations" and "failure to follow instructions", whereas larger LMs like GPTs and Claude are more prone to "improper function calls" and "inaccurate queries". Notably, GPTs struggle more with SQL generation, while Claude performs better in that area but has higher rates of improper function usage. For a more comprehensive discussion of these error patterns, specific cases, and proposed strategies for addressing them, refer to Appendix H.

Table 4: Ablation study of different proposed components in SheetAgent on SheetRM dataset.

Method	Exec@1↑	Pass@1↑	SubPass@1↑
SheetAgent (GPT-3.5)	92.4	**31.2**	**69.8**
-*w/o* Informer	**95.3**(+2.9)	13.2(-18.0)	65.8(-4.0)
-*w/o* Retriever	84.2(-8.2)	19.9(-11.3)	63.7(-6.1)
-*w/o* Informer+Retriever	88.6(-3.8)	11.4(-19.8)	57.5(-12.3)

Table 5: Ablation study on different representations. Best results are bolded and suboptimal results are underlined.

Representation	WTQ	SheetRM		
		Exec@1↑	Pass@1↑	SubPass@1↑
JSON	**63.3**	**92.4**	**31.2**	**69.8**
DFLoader	59.7	<u>90.9</u>	<u>30.0</u>	<u>67.5</u>
Markdown	58.6	90.2	29.0	65.4
HTML	<u>62.1</u>	85.2	23.3	57.5

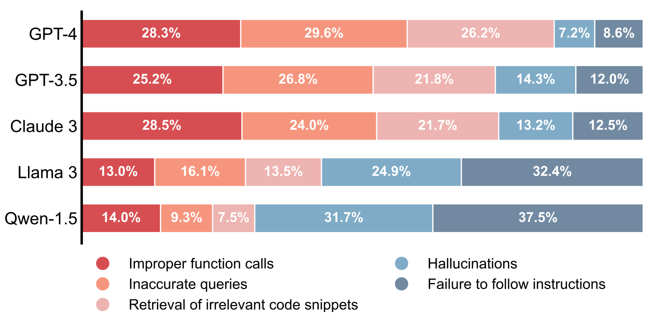


Figure 6: Distributions of different failures cases for various LLM backbones on the SheetRM dataset.

An Illustrative Case Between SheetAgent and SheetCopilot. Figure 7 presents a case with reasoning challenge. For SheetAgent, the Informer accurately selects the key evidence related to the task instruction. The Planner correctly fulfill the task based on the evidence. In contrast, SheetCopilot merely offers a rigid solution that fails to complete the task despite its successful execution.

5 Related Work

LLMs for Table Reasoning. Recent research [35, 36] has demonstrated the excellent ability of LLMs for table reasoning tasks. Chen [5] showcased that LLMs like GPT-3 [2] are capable of reasoning over tables. Binder [8] leverages Codex [4] to generate executable SQL programs to answer table-based questions. DATER [39] decomposes the table and question into finer granularity descriptions through Codex. StructGPT [17] designs an LLM-based framework for structured data and uses it for table question answering. However, these methods are tailored for tasks like question answering or fact verification, typically involving direct queries or explicit statements. As a result, they struggle to handle long-horizon manipulation tasks because of dynamic changes and token limits.

Automatic Spreadsheet Manipulation. Early research [1, 11–13] focus on leveraging program synthesis to guide spreadsheet manipulation. However, these methods fail to generate effective programs without high-quality query specifications. To address

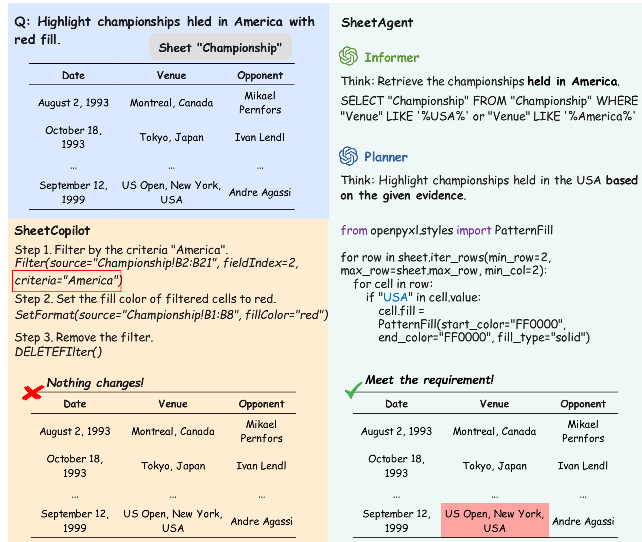


Figure 7: A comparison between SheetAgent and SheetCopilot on a task with reasoning challenges. SheetCopilot generates a rigid solution that fails to fulfill the task. SheetAgent identifies the task intention and gives a correct solution.

this, some work [7, 9, 15, 31] employ deep learning methods to automate spreadsheet manipulation tasks. Despite excellent performance in narrow domains like formatting and formula prediction, they cannot handle a broader range of operations. Given the remarkable performance of LLMs on various tasks [32, 33, 41], their use for comprehensive spreadsheet manipulation has been explored [20, 26, 37, 40, 42]. Payan et al. [26] utilizes LLMs to generate OfficeScripts code with multiple domains. SheetCopilot [20] builds an autonomous agent for invoking custom APIs to manipulate spreadsheets. TableGPT [40] fine-tunes an LLM to understand and operate on tables using external functions. OS-Copilot [37] proposes an OS-oriented agent framework that automates spreadsheet manipulation. However, they simplify real-world requirements and ignore reasoning challenges like unclear expression and multi-step reasoning. Unlike them, we further explore real-life reasoning challenges and propose a collaborative agent framework to solve spreadsheet manipulation tasks that involve these reasoning challenges.

6 Conclusion

In this work, we introduce SheetRM, a more complex and realistic benchmark designed to evaluate the capabilities of agents in performing precise spreadsheet manipulations that require advanced reasoning abilities. Furthermore, We introduce SheetAgent that leverages the power of LLMs to tackle these challenging tasks. Comprehensive experiments have been conducted to assess the reasoning and manipulation proficiency of SheetAgent. We anticipate that SheetRM will serve as a cornerstone for advancing the development of sophisticated generalist agents dedicated to spreadsheet tasks. Furthermore, we hope SheetAgent can alleviate the burden of tedious sheet transactions through automated workflows. While SheetAgent demonstrates strong performance, we acknowledge several limitations like library coverage and token usage, detailed in Appendix J, which we leave as future work.

References

[1] Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. 2016. DeepCoder: Learning to Write Programs. In *International Conference on Learning Representations*.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[3] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* 15, 3 (2024), 1–45.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[5] Wenhu Chen. 2023. Large Language Models are few (1)-shot Table Reasoners. In *Findings of the Association for Computational Linguistics: EACL 2023*. 1090–1100.

[6] Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. TabFact: A Large-scale Dataset for Table-based Fact Verification. In *International Conference on Learning Representations*.

[7] Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. 2021. Spreadsheetcoder: Formula prediction from semi-structured context. In *International Conference on Machine Learning*. PMLR, 1661–1672.

[8] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2022. Binding Language Models in Symbolic Languages. In *The Eleventh International Conference on Learning Representations*.

[9] Haoyu Dong, Jiong Yang, Shi Han, and Dongmei Zhang. 2020. Learning Formatting Style Transfer and Structure Extraction for Spreadsheet Tables with a Hybrid Neural Network Architecture. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2389–2396.

[10] Alexander Edeling, Shuba Srinivasan, and Dominique M Hanssens. 2021. The marketing–finance interface: A new integrative review of metrics, methods, and findings and an agenda for future research. *International Journal of Research in Marketing* 38, 4 (2021), 857–876.

[11] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices* 46, 1 (2011), 317–330.

[12] Sumit Gulwani, William R Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM* 55, 8 (2012), 97–105.

[13] Sumit Gulwani and Mark Marron. 2014. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 803–814.

[14] Md Morshadul Hasan, József Popp, and Judit Oláh. 2020. Current landscape and influence of big data on finance. *Journal of Big Data* 7, 1 (2020), 1–17.

[15] Wanrong He, Haoyu Dong, Yihuai Gao, Zhichao Fan, Xingzhuo Guo, Zhitao Hou, Xiao Lv, Ran Jia, Shi Han, and Dongmei Zhang. 2023. HermEs: Interactive Spreadsheet Formula Prediction via Hierarchical Formulet Expansion. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 8356–8372.

[16] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Mueller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly Supervised Table Parsing via Pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 4320–4333.

[17] Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 9237–9251.

[18] Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. OmniTab: Pretraining with Natural and Synthetic Data for Few-shot Table-based Question Answering. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 932–942.

[19] Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2012. Enterprise data analysis and visualization: An interview study. *IEEE transactions on visualization and computer graphics* 18, 12 (2012), 2917–2926.

[20] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and ZHAO-XIANG ZHANG. 2024. SheetCopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems* 36 (2024).

[21] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263* (2023).

[22] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. TAPEX: Table Pre-training via Learning a Neural SQL Executor. In *International Conference on Learning Representations*.

[23] Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, et al. 2022. FeTaQA: Free-form Table Question Answering. *Transactions of the Association for Computational Linguistics* 10 (2022), 35–49.

[24] Vaishali Pal, Evangelos Kanoulas, and Maarten Rijke. 2022. Parameter-Efficient Abstractive Question Answering over Tables or Text. In *Proceedings of the Second DialDoc Workshop on Document-grounded Dialogue and Conversational Question Answering*. 41–53.

[25] Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 1470–1480.

[26] Justin Payan, Swaroop Mishra, Mukul Singh, Carina Negreanu, Christian Poelitz, Chitta Baral, Subhro Roy, Rasika Chakravarthy, Benjamin Van Durme, and Elnaz Nouri. 2023. InstructExcel: A Benchmark for Natural Language Instruction in Excel. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 4026–4043.

[27] Matt Post. 2018. A Call for Clarity in Reporting BLEU Scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*. 186–191.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[29] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. (2019).

[30] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[31] Mukul Singh, José Cambronero, Sumit Gulwani, Vu Le, Carina Negreanu, Elnaz Nouri, Mohammad Raza, and Gust Verbruggen. 2023. FormaT5: Abstention and Examples for Conditional Table Formatting with Natural Language. *Proceedings of the VLDB Endowment* 17, 3 (2023), 497–510.

[32] Hongda Sun, Yuxuan Liu, Chengwei Wu, Haiyu Yan, Cheng Tai, Xin Gao, Shuo Shang, and Rui Yan. 2024. Harnessing Multi-Role Capabilities of Large Language Models for Open-Domain Question Answering. In *Proceedings of the ACM on Web Conference 2024*. 4372–4382.

[33] Bo Wang, Jing Ma, Hongzhan Lin, Zhiwei Yang, Ruichao Yang, Yuan Tian, and Yi Chang. 2024. Explainable Fake News Detection With Large Language Model via Defense Among Competing Wisdom. In *Proceedings of the ACM on Web Conference 2024*. 2452–2463.

[34] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

[35] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *The Eleventh International Conference on Learning Representations*.

[36] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.

[37] Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024. OS-Copilot: Towards Generalist Computer Agents with Self-Improvement. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

[38] Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. 2022. UnifiedSKG: Unifying and Multi-Tasking Structured Knowledge Grounding with Text-to-Text Language Models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. 602–631.

[39] Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. 2023. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 174–184.

[40] Liangyu Zha, Junlin Zhou, Liyao Li, Rui Wang, Qingyi Huang, Saisai Yang, Jing Yuan, Changbao Su, Xiang Li, Aofeng Su, et al. 2023. Tablegpt: Towards unifying tables, nature language and commands into one gpt. *arXiv preprint arXiv:2307.08674* (2023).

[41] Junjie Zhang, Yupeng Hou, Ruobing Xie, Wenqi Sun, Julian McAuley, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. 2024. Agentcf: Collaborative learning with autonomous language agents for recommender systems. In *Proceedings of the ACM on Web Conference 2024*. 3679–3689.

[42] Wenqi Zhang, Yongliang Shen, Weiming Lu, and Yueting Zhuang. 2024. Data-Copilot: Bridging Billions of Data and Humans with Autonomous Workflow. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

A Pseudo Code of SheetAgent

We present the pseudo code of our proposed SheetAgent consisting of the Planner, Informer, and Retriever modules for handling a spreadsheet task. See Alg. 1.

Algorithm 1: SheetAgent Algorithm

Input: $\mathcal{S}$: spreadsheet, I: task instruction, D: spreadsheet description, s: sheet state, θ, ϕ: LMs for Planner and Informer, P^P, P^I: system prompts for Planner and Informer, T: max iteration rounds, K: number of retrieval results, $\mathcal{E}$: embedding model, C_{repo}: code repository

Output: $\hat{\mathcal{S}}$: final processed spreadsheet

db $\leftarrow$ `InitDB`();
$\mathcal{S}_{\text{copy}} \leftarrow$ `Copy`($\mathcal{S}$);
`UpdateDB`(db, $\mathcal{S}_{\text{copy}}$);
`// Initialize iteration counter, action sequence, and planning history`
$t \leftarrow 1; A \leftarrow \emptyset; h \leftarrow \emptyset$;
while $(t \leq T) \wedge (a_t \neq Done)$ **do**
  $q_t \leftarrow$ `Informer`$_\phi(I, P^I, A_{t-1}, s_t)$;
  `// Execute SQL query and get spreadsheet subview`
  $e_t \leftarrow$ `ExecuteSQL`(db, q_t);
  $a_t \leftarrow$ `Planner`$_\theta(I, P^P, D, s_t, h_{t-1}, e_t)$;
  `// Execute code in sandbox and get feedback`
  $o_t \leftarrow$ `Sandbox`(a_t);
  if `Error`(o_t) **then**
    $C_q \leftarrow$ `ExtractErrorCodeSnippet`(a_t);
    $C^K_{ret} \leftarrow$ `Retriever`$(\mathcal{E}, C_q, C_{repo}, K)$;
    `// Re-plan with error information`
    $a_t \leftarrow$ `Planner`$_\theta(I, P^P, D, s_t, h_{t-1}, e_t, o_t, \boldsymbol{C}^K_{ret})$;
  end
  `// Update necessary information`
  $A_t \leftarrow A_{t-1} \cup \{a_t\}$;
  $h_t \leftarrow h_{t-1} \cup \{o_t, a_t\}$;
  $\mathcal{S}_{\text{copy}} \leftarrow$ `UpdateSpreadsheet`$(\mathcal{S}_{\text{copy}}, \{a_t\})$;
  $s_t \leftarrow$ `UpdateDB`(db, $\mathcal{S}_{\text{copy}}$);
  $t \leftarrow t + 1$;
end
`// Apply generated action sequence on original spreadsheet`
$\hat{\mathcal{S}} \leftarrow$ `UpdateSpreadsheet`$(\mathcal{S}, A)$;
return $\hat{\mathcal{S}}$

B Details of SheetRM Benchmark

B.1 Details of Dataset Collection

Spreadsheet Collection. The spreadsheets curated in SheetRM dare derived from an online examination question bank. We filter out files that are protected, corrupted, or otherwise inaccessible. Within each spreadsheet, the first row of each column must include a header, with the actual data entries starting from the second row. Besides, we ensure all data in each sheet begin from cell A1. We assume that these spreadsheets have already undergone a process to remove some personal information. However, to minimize privacy risk by leaking important personal information, we further implement measures to ensure that no privacy issues arise. Specifically, we modify potentially sensitive information, such as adding noise to the age data and anonymizing bookstore names to general labels like Bookstore A, Bookstore B, etc.

Task Verification. As mentioned in Section 2.2, we instruct GPT-4 to generate realistic tasks that mimic user requests adhering to four guidelines: the tasks should only involve predefined operations, cover diverse manipulation categories, exhibit a long-horizon nature by encompassing multiple subtasks, and incorporate at least one subtask that presents the specified reasoning challenges. This procedure yields a collection of 2316 subtasks. We use GPT-3.5 to filter task instructions that have a lot of semantic duplication to maintain uniqueness. After this, 1973 subtasks are reserved. Furthermore, our internal annotators verify these subtasks manually to ensure quality, which increases the probability that they will be completed by LLMs. Specifically, we adopt two strategies: (1) programming and (2) specialized software. For programming, we ask our internal annotators to write code to complete specific subtask. For specialized software, we use Microsoft Excel to solve the subtask. We accept the subtask only if both strategies solve the subtask. This cross-way validation approach guarantees the reliability of the subtasks. We obtain 1625 subtasks after this process. Finally, we combine these subtasks for different spreadsheets considering horizon and complexity, which leads to 317 task instructions.

B.2 Comparison Between SheetRM and SCB

We conclude the differences that highlight the advantages of our proposed SheetRM dataset compared with SCB as follows:

- **More sheets:** The number of spreadsheet files in SheetRM and SCB is comparable. Besides, SheetRM maintains more spreadsheet files and sheets than SCB **(41 vs 28 & 137 vs 31)**. Each spreadsheet file contains more complex logical relationships and information.
- **More subtasks and longer task horizon:** As shown in Table 6, SheetRM maintains more subtasks **(1625 vs 431)** with longer horizon tasks **(averaging 5.13 vs 1.95)**. Detailed task length distribution is presented in Figure 9.
- **Broader categories and more reasonable division:** SCB categorizes tasks into 6 main types: Entry & Manipulation, Formatting, Pivot Tables, Charts, Formulas, and Management, which results in unbalanced coverage and vague definitions. For example, Formula is basically a type of numerical computation and overlaps with Management and Manipulation, etc. In contrast, SheetRM divides the **5 major categories and 36 sub-categories** from coarse to fine and minimizes the overlap of sub-operations. We believe this allows for a better evaluation of the agents.
- **Finer-grained and more flexible evaluation:** We propose an **automated checklist-based evaluation** in SheetRM that is flexible and accurate for each subtask in the middle of a process, whereas SCB directly compares the final spreadsheets with the

ground truth spreadsheets, ignoring the intermediate process of evaluation.

- **Introduction of reasoning challenges:** It is worth noting that reasoning challenges are innovatively introduced combined with manipulation in SheetRM. In real-world spreadsheet tasks, it is often necessary to reason and analyze problems and data in order to carry out operations. The SCB simplifies the task objectives by assessing only the LLM's ability to manipulate spreadsheets. Instead, our proposed SheetRM presents more realistic and challenging tasks. Please refer to Appendix B.4 for further elaboration.

B.3 Detailed Statistics of Dataset

Spreadsheet Files. We provide more detailed statistics of our SheetRM dataset. We collect spreadsheets covering multiple fields. As illustrated in Figure 8 (Left), we categorize these spreadsheet files into five main fields, reflecting the significant areas where spreadsheets are frequently employed to handle a variety of tasks. We manually annotate a short natural language description as a summary for each spreadsheet file, aiming to stimulate inherent knowledge of LLMs. Each description provides an overview for LLMs to better understand the background information. We provide the descriptions in Table 7.

Task Instruction. We cluster the commonly used operation when working with spreadsheets into five categories, namely **Value Processing**, **Worksheet Management**, **Format Adjustment**, **Chart Design**, and **Content Summary**. For each manipulation category, we further break it down into fine-grained operations. We believe these operations can cover most spreadsheet affairs. The description of these operations is introduced in Table 8. Figure 8 (Right) demonstrates the distribution of verb-noun phrases within our 317 task instructions. We highlight the ten most frequent root verbs and their four primary associated nouns, showcasing the diversity of task instructions in the SheetRM dataset. Additionally, we show the distributions of the number of manipulation categories and subtasks for these task instructions (see Figure 9 (Left)). The majority of tasks span 2 or 3 manipulation categories, with a decent portion encompassing 4 categories, underscoring the diversity of tasks in the SheetRM dataset. We further count the number of subtasks in each task. As displayed in Figure 9, each task includes at least 2 sub-tasks, with the most complex extending to 10. Predominantly, the tasks vary in length from 3 to 7. This reflects the long horizon feature of SheetRM, which poses a significant challenge to LLMs. Full prompts for task generation are available in Appendix I.1.

B.4 Explanation of Reasoning Challenges

Our SheetRM dataset stands out from other spreadsheet manipulation collections due to its emphasis on reasoning-dependent manipulation. Specifically, each task incorporates reasoning challenges. We draw inspiration from several popular table reasoning tasks, including table question answering datasets WikiTableQuestions and FeTaQA, and table fact verification task TabFact. We analyze cases within these datasets that most models struggled with and identify four types of reasoning challenges, namely **Complex Computational Logic**, **Vague Requirements**, **Incoherent Data Format**, and **Information Extraction**. We find that these reasoning challenges are prevalent in real-world spreadsheet manipulation tasks due to the diversity of human expression. Thus, integrating practical insights, we incorporate these reasoning challenges into our spreadsheet manipulation tasks. We elaborate these challenges with descriptions and specific examples:

Complex Computational Logic

Description:
Problems that require more than one reasoning steps to be solved.
Example Sheet:

Name	Date of Entry	Educational Qualification	Salary
Alice	3/1/2001	Master	11,100
Bob	12/1/2006	Bachelor	10,350
...	...	...	...
John	1/9/2011	Doctor	41,100

Instruction:
Which period, 2001-2006 or 2007-2012, had a higher proportion of employees with bachelor's degrees? For the period with the higher proportion, calculate the average salary of the undergraduate employees and put it in cell E1.
Challenge:
To fulfill this instruction, the capability of multi-step reasoning is required.

Vague Requirements

Description:
Problems that refer to incomplete or ambiguous specifications which lack clarity and precision, making it challenging to understand and fulfill the intended goals or objectives.
Example Sheet:

BookID	Book Name	Unit Price
BK-83024	VB Programming	38
BK-83026	Access Programming	35
...	...	...
BK-83029	Network Technology	43

Instruction:
Highlight database-related books in yellow.
Challenge:
To fulfill this instruction, Reasoning over the sheet contents to identify which books are relevant to the database.

Table 6: Comparison of statistical data between SheetRM and SCB.

Dataset Name	# Files	# Sheets	# Task Instructions	# Subtasks	Avg. of Task Length	Median of Task Length	Max Task Length
SheetRM (Ours)	**41**	**137**	**317**	**1625**	**5.13**	**5**	**10**
SCB	28	31	221	431	1.95	2	7

Incoherent Data Format

Description:
Problems that arise when the description provided pertains to the spreadsheet data, yet the units or formats mentioned do not align with those represented in the spreadsheet.
Example Sheet:

Name	Date of Birth
Alice	12/27/1964
Bob	9/28/1974
...	...
John	7/19/1987

Instruction:
Mark the names of employees born after 1985-1-1 in red.
Challenge:
To fulfill this instruction, the "Date of Birth" column should be inferred to align the format.

Information Extraction

Description:
Problems that require specific information to be extracted from the spreadsheet.
Example Sheet:

Venue	Opponent	Final Score
Memphis, Tennessee, USA	Jim Courier	7-5, 6-7, 6-7
Australian Open, Melbourne, Australia	Pete Sampras	6-7, 4-6, 4-6
...	...	...
Estoril, Portugal	Albert Costa	6-2, 3-6

Instruction:
Extract the scores from the first round of the finals into the new column "First Round Score".
Challenge:
To fulfill this instruction, Information about the "Final Score" is required to determine how to extract the first round score.

C Explanations of The Code-Centric Design in Planner

What is generated by the Planner is crucial for precise manipulation. Li et al. [20] introduces a set of virtual APIs as the action space for its proposed agent. However, these APIs lack scalability and are prone to hallucinations when invoked due to conflicts with the inherent knowledge of LLMs. Considering the strong code generation capabilities of LLMs [3], we assign the Planner to generate codes to control spreadsheets. During the process of dataset construction (Section 2.2), we find that Python, compared with VBA, is suitable for manipulating spreadsheets and aligns well with existing training corpus [4, 30] for LLMs. As shown in Table 9, we assess various Python libraries for spreadsheet manipulation. Morden software features considers support for newer spreadsheet software functionalities like complex formulas, charts, and conditional formatting. LLM familiarity measures how extensively language models like me can understand, explain, and generate code examples using these libraries. We select a few natural language description and code snippets written by these libraries, prompt several LLMs (`gpt-3.5-turbo-1106`, `qwen-14b-chat`, etc.) to generate and explain code examples, and evaluate the results manually. During this process, we found that *xlwings* code can hardly be understood by these LLMs despite its strengths in other aspects. Finally, We choose to primarily use *openpyxl*[3] and *pandas*[4] as a combination of them can cover all operations shown in Figure 2.

D Details of Code Collection for the Retriever

The Retriever's code comes from GitHub open-source projects and external Python libraries like *openpyxl* and *pandas*, focusing on high-quality, popular code to ensure data representativeness. We organized these codes by operations covered in SheetRM and then abstracted them for universality. For operations without existing code, we gathered more from the same sources or wrote code ourselves, ensuring coverage of all defined operations. The organized code is related to the corresponding tasks, mainly demonstrating the application programming interfaces and providing high-level guidance. However, the specific implementations of these APIs and the generated solutions are different. Since we abstracted and encapsulated the collected codes, we only provided information on how to operate in it, while the application of the actual data is relevant to the task scenario. Thus, we anticipate that LLMs learn from the knowledge provided by these code snippets and reflect on past trajectories to generate more robust and higher-quality solutions to the task. We provide several code examples in Listing 1-2.

E Dataset Details

The details of datasets mentioned in Section 4.1 are provides as follows:

- **WikiTableQuestions** includes intricate questions created by crowd workers from Wikipedia tables. These questions necessitate multiple advanced operations like comparison, aggregation, and arithmetic, demanding a detailed compositional analysis of table entries. This dataset uses CC-BY-SA-4.0 license.
- **FeTaQA** features free-form questions derived from tables that call for profound reasoning and comprehension. Predominantly,

[3] https://openpyxl.readthedocs.io
[4] https://pandas.pydata.org

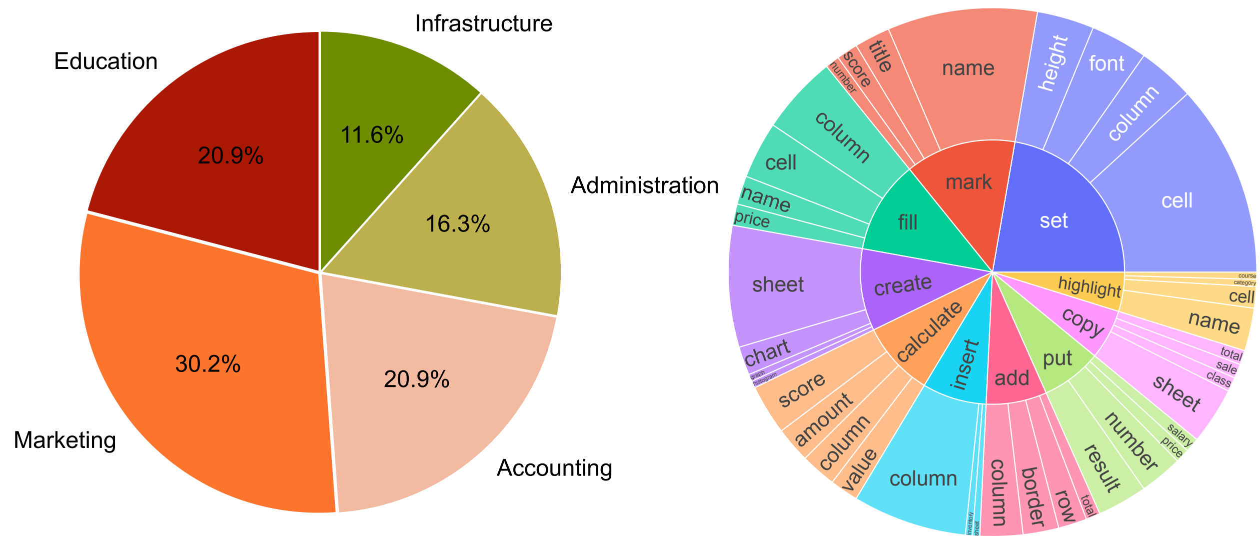


Figure 8: (Left) Distribution of fields to which the spreadsheet files belong. (Right) An illustration of verb-noun phrases in the task instructions. We count the top 10 most frequent root verbs and their associated nouns, ranking the top four for each. These verb-noun combinations showcases the diversity of the generated instructions.

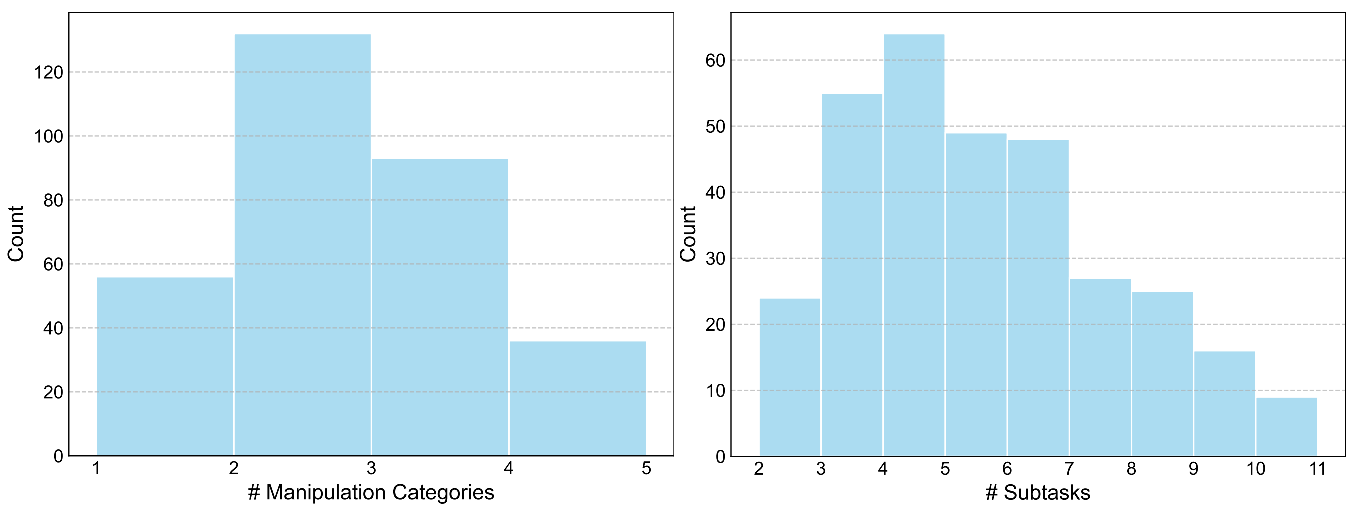


Figure 9: (Left) Distribution of manipulation categories in each task. (Right) Histogram of the task length.

the questions in FetaQA arise from non-contiguous segments of the table. The performance is measured by accuracy on a test set of 2,003 samples. This dataset uses CC-BY-SA-4.0 license.

- **TabFact** serves as a benchmark for table-based fact verification, with crowd workers composing statements from Wikipedia tables. For instance, the statement: "The industrial and commercial panel has four more members than the cultural and educational panel." must be validated as "True" or "False" based on the table information. Accuracy is reported on a smaller test set of 2,024 statements across 298 tables. It uses MIT license.
- **SCB** is a spreadsheet manipulation dataset, which contains 28 spreadsheets collected the Internet. The 221 spreadsheet control tasks within this dataset are generated using GPT-4, including analyzing sales data, calculating financial metrics, and visualizing data with charts. It uses GPL-3.0 license.
- **SheetRM** introduced in Section 2 comprises of 317 real-world spreadsheet tasks. Each task includes multi-category and long-horizon manipulation sequences, along with a specific reasoning challenge. This dataset comprehensively assess the reasoning and manipulation capabilities of LLM-based agents. It also supports finer-grained and more flexible automatic evaluation. For more details of our proposed SheetRM benchmark, please refer to Appendix B. Our SheetRM dataset follows the CC-BY-SA-4.0 license.

Table 7: A short natural language description of the spreadsheet files we collect in SheetRM dataset.

Spreadsheet File	Description
BookSales	This workbook presents data related to book sales.
StudentsGrade	This workbook is about organizing and analyzing student transcripts for first-grade students.
ABProductSales	This workbook presents data related to product A and B.
Reimbursement	The workbook shows the company's travel expense reimbursement status for the year 2013.
ElectronicsSales	The workbook is about conducting statistical analysis of the company's sales.
PayrollSummary	The workbook is the March 2014 employee salary sheet.
TeachingFees	This workbook shows the teaching situation and instructor hourly fees for the courses in the Teaching Research Office in the year 2012.
Deposit	The workbook is a bank deposit journal.
ComputerBookSales	The workbook depicts the sales figures for computer-related books in December 2012.
ScienceMajorGrade	The workbook shows the final exam grades for the Information and Science major.
PersonnalInformation	This workbook is the personnel file information of company employees.
ComputerBookSales2	This workbook represents the sales statistics of computer-related books.
AppliancesSales	This workbook shows the sales statistics of various household appliances.
DepartmentSales	This workbook documents the sales performance of company's products in the first half of the year.
QuartersSales	This workbook summarizes the sales performance for the first two quarters.
FinalGrade	This workbook provides a detailed analysis of students' final grades.
ParkingFees	This workbook keeps track of parking fees and the associated rates.
LivingCosts	This workbook displays an individual's monthly expense report.
StudentsGrade2	This workbook displays the grades for each subject in the class.
LawMajorGrade	This workbook presents the final grade analysis of law students from the 2012 cohort.
YearsSales	This workbook documents the sales statistics of company products in 2012 and 2013.
YearEndSalary	This workbook provides the year-end salary details of employed staff members.
AirQuality	This workbook illustrates the air quality data for major cities in China.
SalesAndPurchase	This workbook is a record of this year's sales and purchase data.
PersonnelChange	This workbook contains the personal details of company employees for the year 2019, including their entry and departure information.
ProductLaunchPlan	This workbook outlines the product launch timeline, key milestones, and marketing strategies.
StudentAttendance	This workbook tracks the attendance records of students across various grades.
QuarterlyEarnings	This workbook presents the company's earnings and financial reports for each quarter of 2020.
OfficeInventory	This workbook lists office supplies, including current stock levels and reorder statuses.
RoadMaintenanceLog	This workbook logs the maintenance schedule and costs associated with road repairs in the city.
CustomerSurvey	This workbook compiles customer feedback from recent marketing campaigns and product surveys.
TeacherPerformance	This workbook evaluates teacher performance based on student feedback and exam results.
BudgetForecast	This workbook forecasts the company's budget allocations for the next fiscal year.
HRLeaveTracker	This workbook tracks employee leave, including vacation days and sick leave balances.
BridgeInspection	This workbook contains the results of bridge safety inspections conducted in 2021.
CampaignROI	This workbook analyzes the return on investment (ROI) of various marketing campaigns.
CourseEnrollments	This workbook tracks student enrollment numbers for various courses during the academic year.
TaxFilingSummary	This workbook summarizes the company's tax filings for the past three years.
MeetingMinutes	This workbook records the minutes and action items from weekly department meetings.
PowerGridStatus	This workbook monitors the status of the city's power grid, including outages and repairs.
AdBudgetAllocation	This workbook details the allocation of the advertising budget across different channels.

F Implementation Details

Baselines. As for table reasoning tasks, we run Binder and DATER using the official implementations. The only difference is that we revise the code to use publicly available *gpt-3.5-turbo-16k-0613* as the LLM backbone instead of Codex due to its inaccessibility. We also run StructGPT on TabFact small-test set and FeTaQA using its open-sourced code with the same LLM backbone. On the proposed SheetRM, we have improved SheetCopilot based on the simplified open-source version[5] with error feedback functionality for fair

[5]https://github.com/BraveGroup/SheetCopilot.

Table 8: Description of each fine-grained operation involved in SheetRM dataset.

Manipulation Category	Operation	Description
Value Processing	Calculate	Calculations and statistics.
	Insert	Insert rows or columns.
	Delete	Delete cells, rows or columns.
	Auto Fill	Fill according to the control relationship.
	Sort	Sort rows or columns in ascending or descending order.
	Copy & Paste	Copy and paste cell values.
	Replace	Replace the values of a cell at a specified location.
	Hyperlink	Set up hyperlinks.
	Distinction	Remove duplicates.
	Filter	Filter specified cells according to certain conditions.
Worksheet Management	Create Worksheet	Create a new worksheet.
	Delete Worksheet	Delete the specified worksheet.
	Rename Worksheet	Rename the specified worksheet.
	Label Color	Modify the color of worksheet name labels.
	Page Size	Modify page size.
	Orientation	Set the page orientation.
Format Adjustment	Font Name	Set the font category.
	Font Color	Set the font color.
	Font Size	Set the font size.
	Bold & Italic	Set the font to be bold or slanted.
	Underline	Underline cell contents.
	Merge & Unmerge	Merge or split cells.
	Alignment	Align cells horizontally or vertically.
	Row Height & Column Width	Set cell row height or column width.
	Background Fill	Set cell background fill color.
	Numeric Format	Set cell number formatting.
Chart Design	Chart Type	Set the Chart Type.
	Chart Data Source	Set the data source for the chart.
	Chart Caption	Set the title of the chart.
	Chart Legend	Set the Chart Legend.
	Chart Position	Specify where to place the chart.
	Chart Axis	Set the axes of a chart.
	Data Label	Set data labels for charts.
	Trendline	Add a trendline to the chart.
Content Summary	Pivot Creation	Create pivot table.
	Summary Function	Set statistical functions of the pivot.

Table 9: Comparison of Python libraries for spreadsheet manipuation.

Library	Read	Write	Additional Features	Cross-Platform	Modern Software Features	LLM Familiarity
xlrd	✓	✗	✗	✓	Limited	High
xlwt	✗	✓	✗	✓	Limited	High
openpyxl	✓	✓	✓	✓	High	High
xlwings	✓	✓	✓	✓	High	Low
xlsxwriter	✗	✓	✗	✓	Limited	High

```python
# create_pie_chart.py
from openpyxl import Workbook
from openpyxl.chart import PieChart, Reference

data = [
    ["Pie", "Sold"],
    ["Apple", 50],
    ["Cherry", 30],
    ["Pumpkin", 10],
    ["Chocolate", 40],
]

wb = Workbook()
ws = wb.active

for row in data:
    ws.append(row)

pie = PieChart()
labels = Reference(ws, min_col=1, min_row=2, max_row=5)
data = Reference(ws, min_col=2, min_row=1, max_row=5)
pie.add_data(data, titles_from_data=True)
pie.set_categories(labels)
pie.title = "Pies sold by category"

ws.add_chart(pie, "D1")
```

Listing 1: Implementation for pie chart creation.

comparison. For the VBA method, we adjust the prompt of SheetAgent to generate *pywin32* code and remove the Retriever module due to code repository mismatch. FormaT5 and SpreadsheetCoder are implemented using the official open-sourced code. For the rest baselines, we report the performance obtained from papers.

LLM Backbones for SheetAgent. In the main experiments, we select various LLMs as the backbones for our proposed SheetAgent. As for proprietary LLMs, we choose GPT-3.5, GPT-4[6], and Claude 3 (`claude-3-sonnet-20240229`[7]). In terms of open-source LLMs, we adopt Qwen-1.5 (`qwen-14b-chat`[8]) and Llama 3 (`llama3-8b-instruct`[9]). Note that multiple versions of GPTs are involved for alignment with other baselines. Specifically, for SCB, WikiTableQuestions, FeTaQA, and TabFact, we use `gpt-3.5-turbo-16k-0613`. For our SheetRM, we employ `gpt-3.5-turbo-1106` and `gpt-4-turbo-0409`.

Choice of In-context Examples. For the SCB dataset, we align with SheetCopilot by using only one in-context example. For other

[6]https://platform.openai.com/docs/models
[7]https://docs.anthropic.com/claude/docs/models-overview
[8]https://huggingface.co/Qwen/Qwen-14B-Chat
[9]https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

```python
# add_second_axis.py
from openpyxl import Workbook
from openpyxl.chart import BarChart, LineChart,
↪ Reference

wb = Workbook()
ws = wb.active

rows = [
    ["Aliens", 2, 3, 4, 5, 6, 7],
    ["Humans", 10, 40, 50, 20, 10, 50],
]

for row in rows:
    ws.append(row)

c1 = BarChart()
v1 = Reference(ws, min_col=1, min_row=1, max_col=7)
c1.add_data(v1, titles_from_data=True,
↪ from_rows=True)

c1.x_axis.title = "Days"
c1.y_axis.title = "Aliens"
c1.y_axis.majorGridlines = None
c1.title = "Survey results"

# Create a second chart
c2 = LineChart()
v2 = Reference(ws, min_col=1, min_row=2, max_col=7)
c2.add_data(v2, titles_from_data=True,
↪ from_rows=True)
c2.y_axis.axId = 200
c2.y_axis.title = "Humans"

# Display y-axis of the second chart on the right by
↪ setting it to cross the x-axis at its maximum
c1.y_axis.crosses = "max"
c1 += c2

ws.add_chart(c1, "D4")
```

Listing 2: Implementation for adding second axis in a chart.

datasets, we utilized two in-context examples each. Specifically, for SCB, we selected the same task as SheetCopilot. We initially had VBA and SheetAgent generate a trajectory for the task under a zero-shot setting using GPT-4, then made appropriate modifications to ensure correctness. The modified trajectory was ultimately used as the in-context example. For SheetRM benchmark, we constructed two additional tasks not present in the dataset and employed VBA, SheetAgent and SheetCopilot to generate trajectories for these tasks, in the same manner, to serve as in-context examples. For other datasets, including WTQ, FeTaQA, and TabFact, where our experiments were conducted on the test sets, we chose two tasks from their respective training sets as examples. It is worth noting that our SheetAgent only uses 2 in-context examples while Binder uses 14. SheetAgent still achieves superior performance.

Computing Power. All the results in our experiments are obtained by running the code on a server equipped with an Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz and 2*NVIDIA A800.

Accessibility of Code and Data. Supplementary information is available in the Appendix. Additional information such as code and data is available at https://anonymous.4open.science/r/SheetAgent. We also present a video demo at the project website.

G Additional Experimental Results

G.1 Full Results on Table Reasoning Tasks

Table 10-11 show the full evaluation results on table reasoning tasks.

Table 10: Results of different methods on WTQ test set and TabFact small-test set. We report the accuracy metric. Best results are bolded and second-best results are underlined.

Method	WTQ	TabFact
Fine-tuning based LLMs		
TAPAS [16]	48.8	83.9
TAPEX [22]	57.5	84.2
UnifiedSKG [38]	49.3	**85.4**
OmniTab [18]	<u>62.8</u>	82.8
Prompting based LLMs		
GPT-3 CoT [5]	45.7	76.0
Binder [8]	59.9	82.9
DATER [39]	61.6	80.7
StructGPT [17]	52.2	81.2
SheetAgent (Ours)	**64.4**	<u>84.8</u>

G.2 Vision-Enabled SheetAgent

We have explored the potential of leveraging GPT-4V(ision)'s visual capabilities by substituting spreadsheet snapshots for the text-modal sheet state in the observation. Given the cost of GPT-4V and the challenges in automatic snapshot capture of spreadsheets, we test this approach with 10 representative tasks from the SheetRM dataset. We have ensured these tasks span all five manipulation categories defined in SheetRM. As vision-eanbled SheetAgent can observe full state of spreadsheets, we remove the Informer module for fair comparison. We present the differences in sheet state

Table 11: Results of different methods on FeTaQA test set. Best results are bolded and suboptimal results are underlined.

Method	sacreBLEU
Fine-tuning based LLMs	
T5-small [23]	21.6
T5-base [23]	28.1
T5-large [23]	30.5
TAPEX [22]	34.7
UnifiedSKG [38]	33.4
PeaQA [24]	33.5
OmniTab [18]	<u>34.9</u>
Prompting based LLMs	
GPT-3 CoT [5]	27.0
Binder [8]	31.6
DATER [39]	30.9
StructGPT [17]	32.5
SheetAgent (Ours)	**36.7**

Table 12: Performance comparison between SheetAgent (GPT-4) and SheetAgent (GPT-4V) on 10 representative tasks from SheetRM. Vison-enabled SheetAgent removes the Informer module.

Method	Pass@1 ↑	SubPass@1 ↑
SheetAgent (GPT-4V)	40.0	66.5
SheetAgent (GPT-4)	50.0	74.1

between GPT-4V and GPT-4 in Figure 10. Results are presented in Table 12. Through this intriguing experiment, we observe that when tasks involved visual elements, such as formatting and chart modification, SheetAgent can better adjust styles based on the visual feedback, improving task completion. However, SheetAgent (GPT-4V) has difficulty processing large-scale spreadsheets and correctly identifying sheet data due to low image solution, resulting in task failure. We also obtain an interesting finding that there exists an overlap between the visual aspect of GPT-4V and the Informer module. While GPT-4V allows SheetAgent to perceive multimodal content within spreadsheets (charts, pivot tables, frozen panes, etc.), it faces challenges capturing accurate information in larger-scale tables compared with the Informer. We leave this for our future work.

G.3 Ablation Study on Table Representation

Tabular data is a kind of information-dense structured data, it is crucial to design reliable representations to enable LLMs clearly recognize the attribute relationships. To investigate what representation can better help LLMs to reason over tables. We ablate 4 prevalent table representations: JSON, DFLoader, Markdown, and HTML for SheetAgent on WikiTableQuestions and SheetRM. We provide an illustration of these representations, as shown in Figure 11. Notably, DFLoader is represented by the corresponding

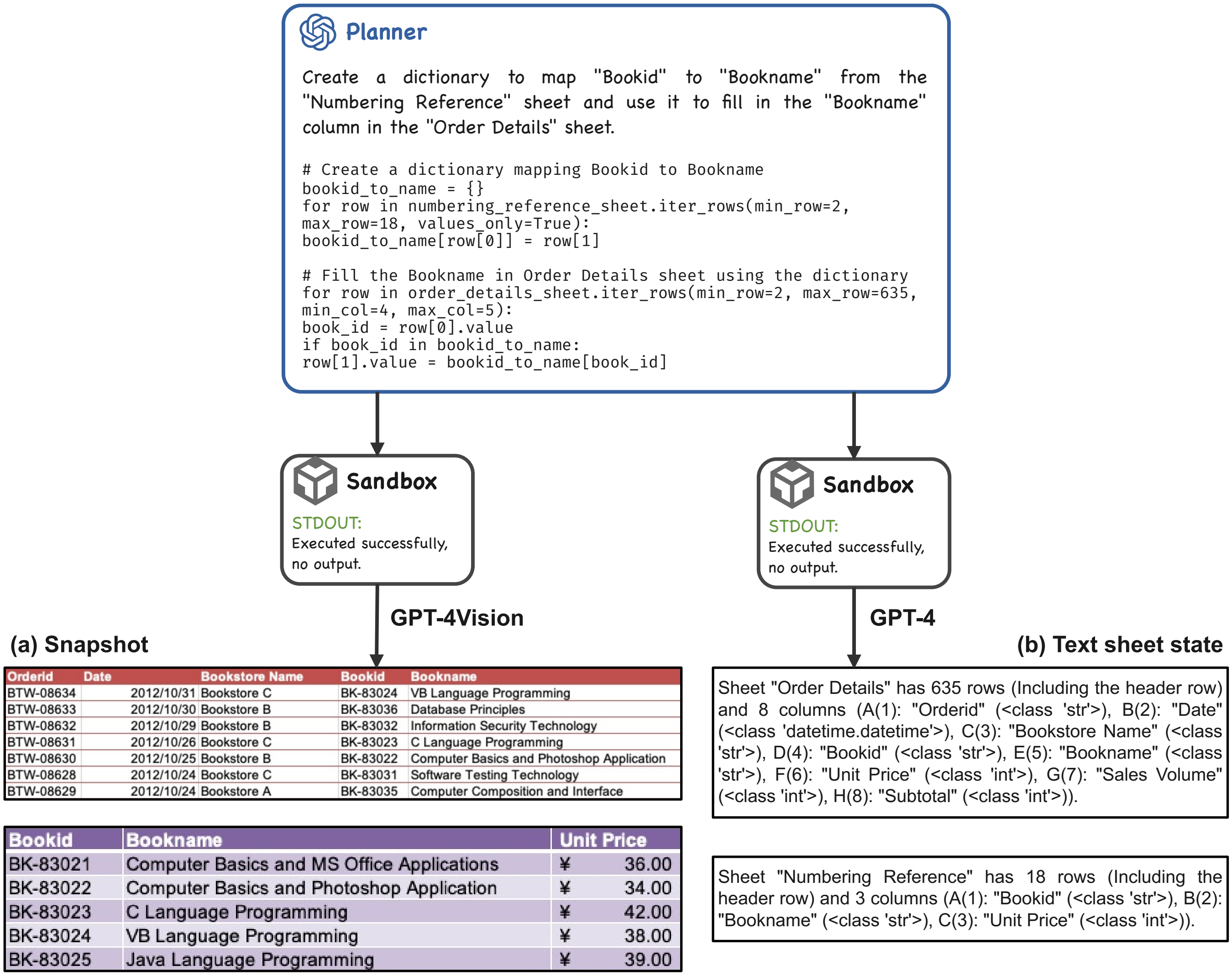


Figure 10: An illustration depicting the differences in sheet state between GPT-4V and GPT-4. For brevity, the Informer and Retriever modules are excluded. The snapshots (namely the visual representation of sheet state) are partial due to the limitation of spreadsheet scale.

Python code snippet that uses the *pandas* DataFrame API to define the table. The results shown in Table 5 reveal that JSON outperform other formats. HTML format achieves a suboptimal result on WTQ, but ranks lowest on SheetRM. Its open-and-close structure helps LLMs understand better, but the verbosity risks exceeding token limits, thus hindering efficiency. We also observe that DFLoader format achieve commendable results, possibly due to its code structure, which might be easier for LLMs to comprehend. Overall, JSON is a preferable choice for both reasoning intensive tasks, like WTQ, and long-horizon tasks with fewer reasoning elements, such as SheetRM.

G.4 Ablation Study on LLM Temperature

We conduct evaluations of our method using the proposed SheetRM dataset under varying conditions by adjusting the temperature settings to investigate the impact of temperature on the performance of LLMs. For these experiments, `gpt-3.5-turbo-1106` is selected as the LLM backbone. Our findings reveal that our method, SheetAgent, achieves its best performance at a temperature of 0.0, with minor performance fluctuations observed at a temperature of 0.2. However, a noticeable decline in performance across all metrics occurs when the temperature is increased to 0.4. This trend suggests that higher temperature settings lead to more unpredictable outcomes from SheetAgent, reflecting a decrease in the stability and reliability of the solutions it generates.

HTML

```html
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
      <th>Occupation</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>John</td>
      <td>28</td>
      <td>Engineer</td>
    </tr>
    <tr>
      <td>Emily</td>
      <td>36</td>
      <td>Teacher</td>
    </tr>

    ...
  </tbody>
</table>
```

Markdown

```
| Name    |   Age | Occupation   |
|:--------|------:|:-------------|
| John    |    28 | Engineer     |
| Emily   |    36 | Teacher      |
| Michael |    29 | Doctor       |
|   ...   |  ...  |     ...      |
```

DFLoader

```python
pd.DataFrame({
    "Name": ["John", "Emily", "Michael", ...],
    "Age": [28, 36, 29, ...],
    "Occupation": ["Engineer", "Teacher", "Doctor", ...]
})
```

JSON

```
{
    {"Name":"John","Age":28,"Occupation":"Engineer"},
    {"Name":"Emily","Age":36,"Occupation":"Teacher"},
    {"Name":"Michael","Age":29,"Occupation":"Doctor"},
     ...
}
```

Figure 11: An illustration of 4 different table representations we use in our experiments.

Table 13: Ablation study on the temperature of LLM.

Temperature	Exec@1↑	Pass@1↑	SubPass@1↑
0.0	92.4	31.2	69.8
0.2	93.5	30.3	68.1
0.4	90.2	28.7	66.2

Table 14: Token and time consumption comparison. Consumption of tokens is calculated by stage.

Method	Avg. # Tokens	Avg. Time (s)
SheetAgent (GPT-3.5)	System Prompt: 324 + Few-shot Demonstrations: 2013 + Planner: 589.7 + Informer: 513.9 + Retriever: 625.3 = **4065.9**	6.9
SheetCopilot (GPT-3.5)	System Prompt: 1895 + Few-shot Demonstrations: 1592 + Plan: 628.5 = 4115.5	**5.8**

G.5 Performance-Cost Analysis

We have conducted extra experiments to calculate the token and time consumption of our method on SheetRM. We use `gpt-3.5-turbo-1106` as the LLM backbone. We compare our proposed SheetAgent with SheetCopilot. It is noteworthy that since SheetCopilot is insufficiently capable of fulfilling a complete task from SheetRM, we select 20 subtasks for which both can generate successful trajectories and calculate metrics based on these. The results are presented in Table 14. On average, our approach consumes fewer tokens compared to SheetCopilot, primarily because SheetCopilot often makes errors, which leads to reflection. However, SheetAgent involves querying multiple LLMs and a vector database, which places us at a disadvantage in terms of time efficiency.

As for cost, we have calculated the cost of successful trajectories by our SheetAgent (GPT-3.5) on SheetRM. The average cost to finish a complete task is $0.0049. Notably, excellent results of SheetAgent shown in Table 2 can be achieved even with relatively cheap backbone GPT-3.5 and Claude 3 Sonnet, which is a trade-off between cost and performance. We believe the superior performance of SheetAgent compared to other methods justify this resource use.

H Failure Cases Analysis

In the main text (Section 4.6), we briefly highlighted the key differences in the error distributions of various LLM backbones. To gain a clearer understanding of the differences between the LLM backbones compared in Sections Section 4.2 and Section 4.3, we conduct a detailed error analysis to determine the causes and locations of failures. We classify the reasons for failure as follows:

- **Improper function calls:** The Planner inaccurately invokes function interfaces for certain operations. For example, it uses `chart.set_title("Chart Title")` instead of `chart.title =`

"Chart Title", resulting in an **AttributeError**. Additionally, it performs worksheet.cell_range() (a method deprecated in newer versions of *openpyxl*) instead of worksheet.iter_rows() or worksheet.iter_cols().

- **Inaccurate queries:** The Informer generates imprecise SQL statements leading to incorrect or incomplete information being input. For example, it attempts to query books related to office software but produces a statement like SELECT * FROM w WHERE BookName LIKE '%Excel%' OR BookName LIKE '%PowerPoint%' while ignoring Word.
- **Retrieval of irrelevant code snippets:** The Retriever sources irrelevant code fragments, which impedes the Planner's correction process. This happens due to similarities in code segments within the code repository, resulting in the retrieval of irrelevant code.
- **Hallucinations:** It performs operations on rows and columns that are out of scope, ignoring the actual data, or creates data that does not exist.
- **Failure to follow instructions:** It terminates tasks prematurely or fails to comply with the given instructions. For example, it only completes a part of subtasks from a long-horizon task or highlights entries in colors not specified by the task.

We present the proportions of different failure cases for various LLM backbones on our SheetRM in Figure 6. Combining the results in Figure 4, we can observe that: (i) LLMs with poorer performance on the benchmark (e.g., llama3-8b-instruct and qwen-14b-chat) have a significantly higher proportion of errors related to "hallucinations" and "failure to follow instructions". In contrast, well-tuned LLMs with extensive parameters, such as GPT-3.5, GPT-4, and Claude 3 Sonnet, have their errors predominantly concentrated in "improper function calls", "inaccurate queries", and "retrieval of irrelevant code snippets". This indicates that advanced LLMs perform better on complex tasks, whereas smaller open-sourced LLMs struggle significantly with understanding and executing instructions. (ii) Among proprietary LLMs, GPT-3.5 and GPT-4 exhibit similar error distributions, with high proportions of errors in "inaccurate queries" and "improper function calls" (29.6% and 28.3% for GPT-4, 26.8% and 25.2% for GPT-3.5, respectively). In contrast, Claude 3 Sonnet shows a different pattern, with a similar proportion of "inaccurate queries" (24.0%) but a relatively higher proportion of "improper function calls" (28.5%). This may reflect that GPTs are adept at generating proficient Python code, while Claude can better understand complicated instructions and translate them into accurate SQLs. (iii) Smaller open-source LLMs, such as llama3-8b-instruct and qwen-14b-chat, display similar error patterns, primarily in "hallucinations" and "failure to follow instructions". Llama3-8b-instruct possesses a "failure to follow instructions" error rate of 32.4%, whereas qwen-14b-chat has a significantly higher rate of 37.5%. This phenomenon may be attributed to their training corpus and model scale.

We further perform a deep analysis of specific failure cases across different LLM backbones, which reveals distinct patterns and challenges. GPT-4 and GPT-3.5 are prone to make errors in generating correct SQLs. After inspecting the specific bad cases, we find that in most cases, they understand the task instruction but generate SQLs that semantically fail to fulfill the task requirements. In other cases, they generate syntactically incorrect SQL statements that cause execution to fail. Differently, Claude 3 Sonnet owns a highest rate of improper function calls but fewer errors of inaccurate queries. It usually calls a function that does not exist or is deprecated, or misunderstands the function usage. For instance, it uses *openpyxl*'s iter_rows() function to iterate through the spreadsheet. The exact code it produces is `for row in ws.iter_rows(min_row=1, max_row=10, max_col="E"):`, where max_col should be an integer instead of a string. Llama3-8b-instruct and qwen-14b-chat share the highest proportions of instruction-following failures and hallucinations, suggesting difficulties in maintaining task context and adhering to long-horizon instructions. We note that there are a large number of incomplete solutions in the llama3-8b-instruct failure case, due in large part to its limited context length of 8K. For qwen-14b-chat, we observe that it can hardly follow the complicated and long-horizon task instructions, and tends to generate irrelevant contents. We assume this may have something to do with its training strategy and corpus.

We have further proposed potential strategies to overcome the proposed failure cases, which may provide insights for future research in this community:

- **Regarding addressing improper function calls,** we found conflicts between the LLM's training corpus on *openpyxl* versions and current versions. Enhancing understandings of library functions through fine-tuning or tool augmentation might mitigate this.
- **For inaccurate queries,** improving model training with diverse SQL examples through fine-tuning and incorporating a validation layer to check queries against database schemas could enhance accuracy.
- **To combat irrelevant code snippet retrieval,** refining the code repository with detailed descriptions of each example's functionality and intended task scenarios could improve retrieval accuracy.
- **For hallucinations and failure to follow instructions,** we attribute these to the model's inherent limitations, noticing a significant increase in these issues on weaker LLM backbones like llama3-8b-instruct and qwen-14b-chat. Switching to a more robust LLM might alleviate these problems. Explicitly managing task progress (e.g., adding a task decomposition module for procedural execution) or incorporating an LLM-driven Critic module (for sanity check on generated solutions) could also partially address these issues.

I Prompts

I.1 Prompt for Subtask Generation

The subtask generation stage involves two aspects, namely generating subtasks with diverse fine-grained operations, and generating subtasks with four reasoning challenges. Figure 12 lists the prompt for the first aspect. To ensure the generation quality, we prompt GPT-4 to choose 4-5 fine-grained operation at a time. To narrow the gap with realistic requirements, we ask GPT-4 to express in a tone of real-life users. Moreover, an in-context example is provided to teach GPT-4. With these prerequisites, GPT-4 can continuously generate diverse and sufficient subtasks. Fig. 13-16 showcase the prompts for generating subtaks with 4 reasoning challenges. Particularly, GPT-4 is prompted under the principle that the generated subtasks should only be solved by reasoning over spreadsheets. This guarantees the existence of reasoning factors in the subtasks to some extent. For the last three challenges, we ask GPT-4 to annotate response with its thinking logic so that we could verify that it makes sense.

I.2 Prompt for Planner

Figure 17 lists the prompt template for the Planner in SheetAgent. The Planner is prompted to mainly use *openpyxl* and *pandas* to manipulate spreadsheets. We also prompt Planner to reason and plan in a ReAct way. It can invoke *Python* tool to interact with a Python sandbox for solution evaluation, and *Answer* tool to submit the answer corresponding to the question.

I.3 Prompt for Informer

The prompt for the Informer is shown in Figure 18. To increase the robustness and reliability of generated SQLs, we provide the Informer with the table schemas of all sheets, along with 3 example rows.

J Limitations and Potential Social Impact

We list the limitations of our proposed SheetAgent as follows:

- **Library limitations:** SheetAgent automates spreadsheet tasks through Python code generation, utilizing libraries like *openpyxl* and *pandas*. Although this has covered a wide range of operations, it is still missing some customizable functionality at the software level. For instance, complex spreadsheet manipulations that involve advanced Excel features such as pivot tables, macros, or specific formatting options are not fully supported. Enhancements in library capabilities or integration with additional tools could address these gaps.
- **High token usage:** Like existing research to automate spreadsheet manipulation [20], SheetAgent inevitably faces higher token usage for long-horizon tasks. This can lead to increased computational costs and slower processing times. Future work will focus on optimizing task descriptions through more efficient prompting techniques or manual refinement to reduce token consumption and improve overall efficiency.

The implementation of SheetAgent has the potential to bring about several positive social impacts. By automating repetitive and time-consuming spreadsheet tasks, SheetAgent can significantly enhance productivity and efficiency in various industries. This can free up human resources for more strategic and creative work, ultimately leading to better utilization of talent and skills. Additionally, SheetAgent can democratize access to advanced data analysis and processing, making these capabilities available to a broader audience, including individuals with limited technical expertise. This democratization can empower more people to leverage data for informed decision-making and innovation.

However, the introduction of SheetAgent might pose negative social impacts. As with any automation technology, there is a risk of job displacement for roles traditionally centered around manual spreadsheet manipulation. This could lead to economic and social challenges for affected individuals. Moreover, the reliance on computational resources for running SheetAgent, especially for large-scale or long-horizon tasks, could contribute to environmental concerns such as increased energy consumption. Addressing these issues requires proactive measures, including reskilling and upskilling programs to help displaced workers transition to new roles and optimizing the efficiency of SheetAgent to minimize its environmental footprint. Ethical considerations must also be prioritized to ensure transparency, fairness, and the safeguarding of user data privacy and security.

Generation of Subtasks

System prompt

Role

As a spreadsheet expert, you have the ability to formulate specific questions for given spreadsheets. These questions are utilized to evaluate the large language model's capabilities to manipualte spreadsheets.

Constraints

1.Choose 4-5 fine-grained operations from the classification below. Use the provided spreadsheet to create tasks, and then merge them into a complete question.
2.Generate an appropriate number of questions each time.
3.Generate questions from the user's perspective, considering elements such as thought process and tone of speech.
4.Simplify the language by focusing only on subproblems composed of fine-grained operations.
5.List the fine-grained operations involved behind each problem. For example, (fine-grained operation：Numeric Format, Auto Fill, Font Color)
6.Make each question more complex and comprehensive.

Fine-grained operations

Here are the fine-grained operations you can choose within the five categories:
A.Value Processing:Calculate,Insert,Delete,Auto Fill,Sort,Copy & Paste,Replace,Hyperlink,Distinction,Filter
B.Worksheet Management:Create Worksheet,Delete Worksheet,Rename Worksheet,Label Color,Page Size,Orientation
C.Format Adjustment:Font Name,Font Color,Font Size,Bold & Italic,Underline,Merge & Unmerge,Alignment,Row Height & Column Width,Background Fill,Numeric Format
D.ChartDesign:Chart Type,Chart Data Source,Chart Caption,Chart Legend,Chart Position,Chart Axis,Data Label,Trendline
E:ContentSummary:Pivot Creation,Summary Function

In-context example

I will give you an example first:
Given a spreadsheet:
Sheet name "Order Details":
Orderid	Date	Bookstore Name	Bookid	Bookname	Unit Price	Sales Volume	Subtotal	Purchaser
BTW-08634	2012/10/31	A	BK-83024	VB Language Programming	38	36	1,368.0	Hongyu Ma
BTW-08633	2012/10/30	B	BK-83036	Database Principles	37	49	1,813.0	Bob
BTW-08632	2012/10/29	C	BK-83032	Information Security Technology	39	20	780.0	Dave

Referring to the details provided in the table above, I'll present the following complex computational logic questions:
1.In sheet "Order Details", adjust "Unit Price" and "Subtotal" to accounting with 2 decimal places and CNY currency symbol format. Fill the "Bookname" column based on "Bookid". Identify the bookstore with the largest decrease in "Subtotal" in January to October 2012 compared to January to October 2011.（fine-grained operation：Numeric Format，Auto Fill, Font Color）
2.In sheet "Order Details", fill in the "BookiD" based on the "Bookname". Add a column named "Adjusted Quantity" to the last column of the sheet. Calculate the data in this column by multiplying 20 with the current values in the "Subtotal" column and then dividing by the maximum value of Subtotal.（fine-grained operation：Auto Fill, Calculate, Shading, Font Color）

User instruction

Now it's your turn.
Given a spreadsheet:
Sheet name "Grade":
Student Number	Name	Class	Chinese	Mathematics	English	Biological	Geography	History	Politics	Total Score	Average Score
120305	Alice	class 3	91.5	89	94	92	91	86	86	629.5	89.93
120203	Charlie	class 2	93	99	92	86	86	73	92	621	88.71
120206	Frank	class 2	100.5	103	104	88	89	78	90	652.5	93.21

GPT response

Sort the students' names alphabetically. Identify the student who has the highest average score after sorting. (fine-grained operation: Sort, Calculate)
Merge cells to group students based on their "Class". Find out which class has the highest total sum of all subjects combined. (fine-grained operation: Merge & Unmerge, Calculate, Sort)
Create a new column called "Science Total", which sums up the scores for "Biological", "Geography", and "History". Then, find the student with the lowest sum in "Science Total". (fine-grained operation: Calculate, Insert)
Adjust the "Total Score" and "Average Score" columns to display only one decimal place. After this adjustment, determine if there are any students whose average score is exactly equal to the rounded "Average Score". (fine-grained operation: Numeric Format, Operation)

Figure 12: Prompt for generating subtasks with diverse fine-grained operations.

Generation of Subtasks with Reasoning Challenges
Complex Computational Logic

System prompt
Role
As a spredsheet expert, you have the ability to formulate specific questions for given spreadsheets. These questions are utilized to evaluate the large language model's comprehension of spreadsheets.

Constraints
1.Begin by grasping the content of the provided worksheets, then formulate questions related to "Complex Computational Logic." These types of questions necessitate the use of multiple statistical procedures for resolution.
2.Keep the focus of the question on asking, without delving into the specifics of the procedural steps.
3.The guiding principle for formulating questions is that they should only be solvable by referencing the content of the worksheets.
4.Generate an appropriate number of questions each time you approach the task.
5.Conclude each question with a fine-grained operation, integrating it with the problem to present the result.
6.Elevate the complexity and comprehensiveness of each question.

Fine-grained operations
Here are the fine-grained operations you can choose within the five categories:
A.Value Processing: Calculate,Insert,Delete,Auto Fill,Sort,Copy & Paste,Replace,Hyperlink,Distinction,Filter
B.Worksheet Management: Create Worksheet,Delete Worksheet,Rename Worksheet,Label Color,Page Size,Orientation
C.Format Adjustment: Font Name,Font Color,Font Size,Bold & Italic,Underline,Merge & Unmerge,Alignment,Row Height & Column Width,Background Fill,Numeric Format
D.ChartDesign: Chart Type,Chart Data Source,Chart Caption,Chart Legend,Chart Position,Chart Axis,Data Label,Trendline
E:ContentSummary: Pivot Creation,Summary Function

In-context example
I will give you an example first:
Given a spreadsheet
Sheet name "Order Details":
Orderid	Date	Bookstore Name	Bookid	Bookname	Unit Price	Sales Volume	Subtotal	Purchaser
BTW-08634	2012/10/31	A	BK-83024	VB Language Programming	38	36	1,368.0	Hongyu Ma
BTW-08633	2012/10/30	B	BK-83036	Database Principles	37	49	1,813.0	Bob
BTW-08632	2012/10/29	C	BK-83032	Information Security Technology	39	20	780.0	Dave

Referring to the details provided in the table above, I'll present the following complex computational logic questions:
1.In sheet "Order Details", identify the quarter with the highest book sales in 2011, then determine which bookstore has the most sales in this quarter, and fill its name of bookstore in B6 cell.（fine-grained operation：Calculate）
2.Identify the bookstore with the largest decrease in "Subtotal" in January to October 2012 compared to January to October 2011. Then, determine the best-selling book with the most volume in that year for this bookstore and highlight its row in red font in the "Order Details" sheet.（fine-grained operation：Font Color）

User instruction
Given a spreadsheet:
Sheet name "Grade":
Student Number	Name	Class	Chinese	Mathematics	English	Biological	Geography	History	Politics	Total Score	Average Score
120305	Alice	class 3	91.5	89	94	92	91	86	86	629.5	89.93
120203	Charlie	class 2	93	99	92	86	86	73	92	621	88.71
120206	Frank	class 2	100.5	103	104	88	89	78	90	652.5	93.21

GPT response
From the "Grade" sheet, calculate the average score of each subject across all classes, identify the subject with the lowest class average, and replace the title of that subject with "Needs Improvement" in cell D1.（fine-grained operation：Replace）
In the "Grade" sheet, determine the student with the highest total score for each class, then create a new worksheet named "Top Students" and copy & paste the information of these students into the new worksheet.（fine-grained operation：Copy & Paste）
On the "Grade" sheet, filter the students who scored above 90 in English and then sort these records in descending order by their Total Score. Once sorted, apply a background fill of light green to these rows.（fine-grained operation：Background Fill）
Within the "Grade" sheet, identify the class with the highest average total score and then merge cells A1 and B1 and label this merged cell with "Best Performing Class".（fine-grained operation：Merge & Unmerge）

Figure 13: Prompt for generating subtasks with "complex computational logic" reasoning challenge.

Generation of Subtasks with Reasoning Challenges
Vague Requirements

System prompt
Role
As a spreadsheet expert, you have the ability to formulate specific questions for given spreadsheets. These questions are utilized to evaluate the large language model's comprehension of spreadsheets.

Constraints
1.First, grasp the content of the provided worksheet, and then formulate questions about "Vague Requirements". Questions about vague requirements involve using synonyms or generic terms to identify statistical objects instead of directly using their real names.
2.Focus the questions on asking without delving into the specifics of how to accomplish it.
3.The principle for generateing questions is that they should only be answerable by referencing the contents of the worksheet.
4.Generate an appropriate number of questions each time.
5.Generate questions from the user's perspective, considering elements such as thought process and tone of speech.
6.Conclude each question with a fine-grained operation, integrating it with the problem to present the result.
7.Specify the referenced object; for instance, when mentioning "Office-related books", it refers to "MS Office Advanced Applications" and "Word Applications."

Fine-grained operations
Here are the fine-grained operations you can choose within the five categories:
A.Value Processing: Calculate,Insert,Delete,Auto Fill,Sort,Copy & Paste,Replace,Hyperlink,Distinction,Filter
B.Worksheet Management:Create Worksheet,Delete Worksheet,Rename Worksheet,Label Color,Page Size,Orientation
C.Format Adjustment:Font Name,Font Color,Font Size,Bold & Italic,Underline,Merge & Unmerge,Alignment,Row Height & Column Width,Background Fill,Numeric Format
D.ChartDesign:Chart Type,Chart Data Source,Chart Caption,Chart Legend,Chart Position,Chart Axis,Data Label,Trendline
E:ContentSummary:Pivot Creation,Summary Function

In-context example
I will give you an example first:
Given a spreadsheet:
Sheet name "Order Details":
Orderid	Date	Bookstore Name	Bookid	Bookname	Unit Price	Sales Volume	Subtotal	Purchaser
BTW-08634	2012/10/31	A	BK-83024	VB Language Programming	38	36	1,368.0	Hongyu Ma
BTW-08633	2012/10/30	B	BK-83036	Database Principles	37	49	1,813.0	Bob
BTW-08632	2012/10/29	C	BK-83032	Information Security Technology	39	20	780.0	Dave

Referring to the details provided in the table above, I'll present the following vague requirements questions:
1.In sheet "Order Details", determine the month with the highest sales volume of programming language-related books("programming language-related books" are synonyms or generic terms for "VB Language Programming" and "C Language Programming") in 2012, and then calculate its proportion to the total sales volume of that month. Put the result in A1 of the new sheet（fine-grained operation：Calculate）
2. In sheet "Order Details", mark the Office-related books("Office-related books" are synonyms or generic terms for "MS Office Advanced Applications" and "Word Applications") in red font color.（fine-grained operation：Font Color）

User instruction
Now it's your turn.
Given a spreadsheet:
Sheet name "Grade":
Student Number	Name	Class	Chinese	Mathematics	English	Biological	Geography	History	Politics	Total Score	Average Score
120305	Alice	class 3	91.5	89	94	92	91	86	86	629.5	89.93
120203	Charlie	class 2	93	99	92	86	86	73	92	621	88.71
120206	Frank	class 2	100.5	103	104	88	89	78	90	652.5	93.21

GPT response
In the "Grade" sheet, figure out which student performed best in language courses (language courses is a generic term for "Chinese" and "English") and put the student's name and average score of the language courses in A1 and B1 of a new worksheet.（fine-grained operation: Calculate, Create Worksheet）
In the "Grade" sheet, determine the average score for science courses (science courses is a generic term for "Biological", "Geography") in class 3, filter the students with average science courses scores below this average and highlight their name in yellow.（fine-grained operation: Calculate, Filter, Background Fill）

Figure 14: Prompt for generating subtasks with "vague requirements" reasoning challenge.

Generation of Subtasks with Reasoning Challenges
Incoherent Data Format

System prompt
Role
As a spreadsheet expert, you have the ability to formulate specific questions for given spreadsheets. These questions are utilized to evaluate the large language model's comprehension of spreadsheets.

Constraints
1.Start by comprehending the content of the provided form, and then generate questions related to "Incoherent Data Format" where the description involves the same meaning but different units or data formats.
2.Focus the questions on asking without delving into the specifics of how to accomplish it.
3.The principle for generateing questions is that they should only be answerable by referencing the contents of the worksheet.
4.Generate an appropriate number of questions each time.
5.Generate questions from the user's perspective, considering elements such as thought process and tone of speech.
6.Conclude each question with a fine-grained operation, integrating it with the problem to present the result.
7.List equivalent data at the end of the question.

Fine-grained operations
Here are the fine-grained operations you can choose within the five categories:
A.Value Processing:Calculate,Insert,Delete,Auto Fill,Sort,Copy & Paste,Replace,Hyperlink,Distinction,Filter
B.Worksheet Management:Create Worksheet,Delete Worksheet,Rename Worksheet,Label Color,Page Size,Orientation
C.Format Adjustment:Font Name,Font Color,Font Size,Bold & Italic,Underline,Merge & Unmerge,Alignment,Row Height & Column Width,Background Fill,Numeric Format
D.ChartDesign:Chart Type,Chart Data Source,Chart Caption,Chart Legend,Chart Position,Chart Axis,Data Label,Trendline
E:ContentSummary:Pivot Creation,Summary Function

In-context example
I will give you an example first:
Given a spreadsheet:
Sheet name "Employee Information":
|Job number| Name|Sex|Section|Educational background|Telephone|Telephone type|Date of birth|
19	Bob	Female	Technical department	Master	12383605517	Cell phone	1980/11/16
20	Charlie	Male	Technical department	Undergraduate course	12733582889	Cell phone	1985/6/28
21	David	Male	Technical department	Master	90870409	Landline	1969/4/24

Referring to the details provided in the table above, I'll present the following incoherent data format questions:
1. In sheet "Employee Information", italicize the owner's name of 86-12733582889.（The phone numbers 86-12733582889 and 12733582889 are equivalent.）（fine-grained operation：Bold & Italic）
2. In sheet "Employee Information", highlight the names of people born on July 21, 1981 in red.（The date July 21, 1981 and 1981/7/21 are equivalent)（fine-grained operation：Font Color）

User instruction
Now it's your turn.
Given a spreadsheet:
Sheet name "Payroll":
Name	Department	Position	Id Number	Gender	Date Of Birth	Age	EducationalBackground	On-Board Time	Length Of Service (years)	Contract Monthly Wages	Monthly Seniority Wage	Basic Monthly Wage
Faythe	Management department	General manager	110108196301020119	Male	1963/1/2	115	Doctor	1981/2/1	34	40000	1700	41700
Grace	Administration department	Secretary	110105198903040128	Female	1989/3/4	115	Junior college	2012/3/1	3	4800	60	4860
Judy	Management department	Manager	310108197712121139	Male	1977/12/12	115	Master	2003/7/1	12	12000	360	12360

GPT response
In the current sheet, apply a green background fill to the cell of the employee who joined the company in February 1981. (The date 'Feb-1981' and '1981/2/1' are equivalent.) (fine-grained operation: Background Fill)
Please adjust the Basic Monthly Wage cell for Chunqiu Ding to display it in thousands. (given that $4860 is equivalent to $4.86K.) (fine-grained operation: Numeric Format)

Figure 15: Prompt for generating subtasks with "incoherent data format" reasoning challenge.

Generation of Subtasks with Reasoning Challenges

Information Extraction

System prompt

Role

As a spreadsheet expert, you have the ability to formulate specific questions for given spreadsheets. These questions are utilized to evaluate the large language model's comprehension of spreadsheets.

Constraints

1.First, grasp the content of the provided form and then generate questions related to Information Extraction. The Information Extraction problem involves extracting additional information from the content in the sheet.
2.Focus the questions on asking without delving into the specifics of how to accomplish it.
3.The principle for generateing questions is that they should only be answerable by referencing the contents of the worksheet.
4.Generate an appropriate number of questions each time.
5.Generate questions from the user's perspective, considering elements such as thought process and tone of speech.
6.Conclude each question with a fine-grained operation, integrating it with the problem to present the result.
7.Provide examples of the extracted sources and content. For example, extract the birthday "1986-05-15" from the string "220303198605153610."

Fine-grained operations

Here are the fine-grained operations you can choose within the five categories:
A.Value Processing:Calculate,Insert,Delete,Auto Fill,Sort,Copy & Paste,Replace,Hyperlink,Distinction,Filter
B.Worksheet Management:Create Worksheet,Delete Worksheet,Rename Worksheet,Label Color,Page Size,Orientation
C.Format Adjustment:Font Name,Font Color,Font Size,Bold & Italic,Underline,Merge & Unmerge,Alignment,Row Height & Column Width,Background Fill,Numeric Format
D.ChartDesign:Chart Type,Chart Data Source,Chart Caption,Chart Legend,Chart Position,Chart Axis,Data Label,Trendline
E:ContentSummary:Pivot Creation,Summary Function

In-context example

I will give you an example first:
Given a spreadsheet:
Sheet name "Statistical Report":
Orderid	Date	Bookstore Name	Bookid	Bookname	Unit Price	Sales Volume	Subtotal	Purchaser	PurchaserID
BTW-08634	2012/10/31	A	BK-83024	VB Language Programming	38	36	1,368.0	Hongyu Ma	211322198509260317
BTW-08633	2012/10/30	B	BK-83036	Database Principles	37	49	1,813.0	Bob	211481198401154411
BTW-08632	2012/10/29	C	BK-83032	Information Security Technology	39	20	780.0	Dave	522324197508045617

Referring to the details provided in the table above, I'll present the following information extraction questions:
1.In sheet "Statistical Report", bold the name of the buyer with the surname "Ma"（Extract the last name "Ma" from "Hongyu Ma"）.（fine-grained operation：Bold & Italic）
2. In sheet "Statistical Report", extract the buyer's date of birth based on the Purchaser ID, create a new column labeled "Birthday," and put the result（Extract the birthday "1986-05-15" from "220303198605153610"）.（fine-grained operation：Font Color）

User instruction

Now it's your turn.
Given a spreadsheet:
Sheet name "Championship":
Outcome	Date	Venue	Surface	Opponent in the final	Score in the final
Runner-up	February 15, 1993	Memphis, Tennessee, USA	Hard (i)	Jim Courier	7-5, 6-7(4-7), 6-7(4-7)
Winner	May 17, 1993	Coral Springs, Florida, USA	Clay	David Wheaton	6-3, 6-4
Runner-up	July 26, 1993	Washington D.C., USA	Hard	Amos Mansdorf	6-7(3-7), 5-7

GPT response

In the provided worksheet, could you filter out and display all matches that have a "Score in the final" that includes a tie-breaker set? (Extract the score "6-7(4-7)" as an example of a tie-breaker set). (fine-grained operation: Filter)
In the workbook, identify all the occasions where the final match was won in straight sets and label these rows with a distinct background color (for example, the final against David Wheaton with the score 6-3, 6-4). (fine-grained operation: Background Fill)

Figure 16: Prompt for generating subtasks with "information extraction" reasoning challenge.

Planner

System prompt

Role

You are a spreadsheet agent and a python expert who can find proper functions to solve complicated spreadsheet-related tasks based on language instructions.

Prerequisites

1. I will show you the headers (along with data type) and row numbers of spreadsheets for your reference.
2. Your partner, "Informer," aids in task completion by providing sheet content represented in {table_representation}, known as "potentially helpful information". This information might be truncated due to token limits, so it's essential to deduce the complete information from what is provided.
3. Please provide step-by-step solutions without explanation.
4. You can use any python library, but when it comes to manipulating spreadsheets, you shoud primarily use the openpyxl and pandas library, which has been already imported as `openpyxl` and `pd`.
5. You should only give one python code snippet at a time. Try not to add comments, and if you must, keep them as concise as possible.
6. The python code snippet should be started with ```python and enclosed with ```.
7. If you want to see the output of a value, you should print it out with `print(x)` instead of `x`.

Response Format Guidance

1. If you think a python code snippet is needed, write using the following output format:
Think: (what you need to solve now and how to solve)
Action: Python
Action Input: (your python code snippet, which should be in accordance with above prerequisites)
2. If you think there is a question to be answered, give your answer using the following format:
Think: (how do you get the answer)
Action: Answer
Action Input: (your answer)
2. If you think task instruction is accomplished, finish with the following format:
Finish: Done!

In context example

...

Instruction

Now it's your turn. This Workbook presents data related to book sales. The workbook is already loaded as `workbook` using openpyxl, you only need to load the sheet(s) you want to use manually. Besides, the workbook will be automatically saved, so you don't need to save it manually.

Sheet state: Sheet "Order Details" has 635 rows (Including the header row) and 8 columns (A(1): "Orderid" (<class 'str'>), B(2): "Date" (<class 'datetime.datetime'>), C(3): "Bookstore Name" (<class 'str'>), D(4): "Bookid" (<class 'str'>), E(5): "Bookname" (<class 'NoneType'>), F(6): "Unit Price" (<class 'int'>), G(7): "Sales Volume" (<class 'int'>), H(8): "Subtotal" (<class 'int'>)). Sheet "Numbering Reference" has 18 rows (Including the header row) and 3 columns (A(1): "Bookid" (<class 'str'>), B(2): "Bookname" (<class 'str'>), C(3): "Unit Price" (<class 'int'>)). Sheet "Statistical Report" has 5 rows (Including the header row) and 2 columns (A(1): "Statistical Item" (<class 'str'>), B(2): "Statistical Value" (<class 'int'>)).

Task instruction: Format the "Order Details" worksheet, adjust "Unit Price" and "Subtotal" to accounting with 2 decimal places and CNY currency symbol format. Fill the "Bookname" column based on Bookid. Identify the bookstore with the largest decrease in sales subtotal in 2012 compared to January to October 2011. Then, determine the best-selling book in that year for this bookstore and highlight its row in red in the "Numbering Reference" worksheet.

Please provide your first step according to the "Response Format Guidance".

Planner response

Think: Apply accounting format with 2 decimal places and CNY currency symbol to the "Unit Price" and "Subtotal" columns in the "Order Details" worksheet.
Action: Python
Action Input:

```python
# Load the "Order Details" sheet
order_details_sheet = workbook["Order Details"]

# Apply accounting format with 2 decimal places and CNY currency symbol to "Unit Price" and "Subtotal" columns
for cell in order_details_sheet["F"][1:] + order_details_sheet["H"][1:]:
    cell.number_format = '"¥"#,##0.00_);[Red]("¥"#,##0.00)'
```

......

Figure 17: A prompt template for the Planner.

Informer

System prompt
You are a table retrieval expert who team up with a spreadsheet agent named "Planner" to accomplish complicated spreadsheet-related tasks. Your partner excel in manipulating spreadsheets. However, some of manipulations require an understanding of specific content of the spreadsheet. Therefore, your role is to retrieve key information from spreadsheets for your partner's reference.

In context example
...

Instruction
Now it's your turn. To make it easier for you to retrieve, all sheets are stored in a SQLite database.
Table schema of "First semester final grades":

```
CREATE TABLE "First semester final grades" (
  "Student Number" INTEGER,
  "Name" TEXT,
  "Class" REAL,
  ...
)
/*
3 example rows:
SELECT * FROM "First semester final grades" LIMIT 3;
{
  "1":{"Student Number":120305,"Name":"Alice","Class":null,"Chinese":91.5,"Mathematics":89,"English":94,"Biological":92,"Geography":91,"History":86,"Politics":86,"Total Score ":null,"Average Score":null},
  "2":{"Student Number":120203,"Name":"Charlie","Class":null,"Chinese":93.0,"Mathematics":99,"English":92,"Biological":86,"Geography":86,"History":73,"Politics":92,"Total Score ":null,"Average Score":null},
  ...
}
*/
```

Task instruction: Calculate the total score and average score of each student, and fill in the "Total Score" and "Average Score" columns respectively. Fill in the "Class" column according to the "Student Number". There are three classes: 1, 2, and 3. Fill the cells with scores of no less than 110 points in the three subjects of Chinese, Mathematics, and English in red respectively, and mark the cells with scores above 95 points in the other four subjects in red font.

Previous completed subtasks of the spreadsheet agent (which is presented from your partner's point of view and is more of your partner's thought while finishing the subtasks):
1. Load the "First semester final grades" sheet and calculate the total and average scores for each student.

Given the task instruction and subtasks your partner has completed, predict what your patner will solve next, and determine what content is most needed by your partner. Write a SQLite select statement to achieve this goal. If there is no need to retrieve information from any spreadsheet, please type "pass".
Based on above, responde in the following format:
Think: (how do you think)
Action: (your SQLite statement, e.g., `SELECT * FROM w WHERE age < 18;`, or "pass")

Informer response
Think: According to the task instruction and completed subtasks, the next step for the Planner is more likely be to fill the "Class" column. The rule for filling "Class" depends on the "Student Number" column, so I should provide the agent with the "Student Number" information.
Action: SELECT "Student Number" FROM "First semester final grades" LIMIT 3;

Figure 18: A prompt template for the Informer.