SheetBrain: A Neuro-Symbolic Agent for Accurate Reasoning over Complex and Large Spreadsheets

Ziwei Wang[1*†], Jiayuan Su[2*†], Mengyu Zhou[5‡], Huaxing Zeng[3*], Mengni Jia[4*], Xiao Lv[5], Haoyu Dong[5], Xiaojun Ma[5], Shi Han[5], Dongmei Zhang[5]

[1] Carnegie Mellon University
[2] Zhejiang University
[3] Brown University
[4] University of Cambridge
[5] Microsoft Research

Abstract

Understanding and reasoning over complex spreadsheets remain fundamental challenges for large language models (LLMs), which often struggle with accurately capturing the complex structure of tables and ensuring reasoning correctness. In this work, we propose SheetBrain, a multi-agent reasoning framework featuring a symbolic dataflow designed for accurate reasoning over tabular data, supporting both spreadsheet question answering and manipulation tasks. SheetBrain comprises three core modules: an understanding module, which produces a comprehensive overview of the spreadsheet—including sheet summary and query-based problem insight to guide reasoning; an execution module, which integrates a Python sandbox with preloaded table-processing libraries and an Excel helper toolkit for effective multi-turn reasoning; and a validation module, which verifies the correctness of reasoning and answers, triggering re-execution when necessary. We evaluate SheetBrain on multiple public tabular QA and manipulation benchmarks, and introduce SheetBench, a new benchmark targeting large, multi-table, and structurally complex spreadsheets. Experimental results show that SheetBrain significantly improves accuracy on both existing benchmarks and the more challenging scenarios presented in SheetBench.

Code — https://github.com/microsoft/SheetBrain

Introduction

Large language models (LLMs) have achieved remarkable success across diverse natural language understanding and generation tasks, particularly in domains involving unstructured text (Xu et al. 2025; Wang et al. 2024; Pan et al. 2024). However, their ability to understand and reason over complex spreadsheet data for question answering (QA) and manipulation remains a fundamental challenge (Zhu et al. 2025; Chen et al. 2024; Li et al. 2023). Real-world spreadsheets—especially Excel-like files—often contain multi-table layouts, hierarchical structures, lengthy content, and other complex formatting elements (Wu et al. 2025b; Li et al. 2024b; Zhao et al. 2022; Cheng et al. 2021). These factors significantly increase the difficulty of accurately capturing the table semantics and performing reliable reasoning, limiting the effectiveness of current LLM-based approaches.

Model	Complex	Multi	Large	Sheet Edit	Total
Test Cases	21	20	20	8	69
Vanilla LLM:					
gpt-4.1	16	17	1	–	34
Proprietary Agents:					
BizChat Analyst	18	14	9	6	49
ChatGPT (4o)	13	8	9	–	34
Open-Sourced Agents:					
StructGPT (4.1)	12	1	0	–	13
SheetAgent (4.1)	11	10	10	4	35
SheetBrain (4.1, ours)	**20**	**18**	**11**	**6**	**55**

Table 1: Comparison results on our proposed benchmark *Sheetbench*. Reasoning over complex, multi-table, and large spreadsheets remains a significant challenge for existing LLM-based approaches.

Prior works (Chen et al. 2024; Jiang et al. 2023; Li et al. 2023) have shown that LLMs can perform QA and manipulation effectively on simple tables and handle basic instructions. However, they often struggle with challenging scenarios involving complex and large spreadsheets. This is empirically demonstrated in the examples summarized in Table 1, where comparison results confirm that LLM-based methods still have significant room for improvement. From these cases, we identify three key findings: (1) These methods do not explicitly analyze the spreadsheet structure or understand the query context beforehand, so their reasoning tends to be blind and inefficient; (2) The proprietary agent BizChat (Microsoft Corporation 2025) shows better performance on complex and large tables compared to vanilla LLMs by integrating symbolic code execution sandboxes;

*Work done during internship at Microsoft.
†Ziwei and Jiayuan contributed equally to this work.
‡Corresponding author (mengyu.chou@gmail.com).

(3) The agents often rely on multi-step reasoning that is prone to getting stuck in local or narrow perspectives, lacking mechanisms for reflection.

Inspired by these findings, we present SheetBrain, a reasoning framework that enhances LLMs' ability to understand and reason over complex spreadsheets for QA and manipulation. As shown in Figure 1, SheetBrain begins with an understanding module that generates a comprehensive overview of the spreadsheet, including a sheet summary and query-specific problem insights, which serve as a scaffold for high-level reasoning. Next, an execution module leverages a specialized Python sandbox integrated with preloaded table-processing libraries and a custom Excel helper toolkit to enable tool-augmented, multi-turn reasoning of tabular data. Finally, a validation module verifies the correctness of both reasoning and output, triggering re-execution with improvement feedback when necessary to avoid getting stuck in local errors and to improve reliability. These components bridge the gap between challenging tabular cases and reasoning capabilities of LLM agents, enabling more robust, accurate, and interpretable performance across diverse spreadsheet scenarios.

To evaluate the effectiveness of SheetBrain, we conduct extensive experiments on several public spreadsheet QA and manipulation datasets. Additionally, we introduce SheetBench, a new benchmark specifically curated to reflect challenges such as large sheets, multi-table layouts, and deeply nested structures. Experimental results show that SheetBrain not only outperforms strong baselines—including vanilla LLMs and spreadsheet LLM-based agents—across public benchmarks, but also generalizes significantly better to the complex scenarios presented in SheetBench.

We summarize our contribution as follows:

- We propose **SheetBrain**, a novel spreadsheet agent based on an understand-execute-validate pipeline. It first generates a comprehensive overview of the spreadsheet, then passes it to a neuro-symbolic workflow execution module that integrates the code sandbox along with dedicated libraries and toolkits for reasoning, and finally to a validation module for verification, triggering re-execution with improvement feedback when necessary.
- We introduce **SheetBench**, a new benchmark comprising 69 challenging real-world spreadsheet QA and manipulation tasks, specifically designed to evaluate LLM performance on large-scale, structurally complex spreadsheets featuring multi-table layouts. Experimental results show that SheetBrain achieves state-of-the-art performance, outperforming vanilla LLMs such as GPT-4.1 (OpenAI 2025a), o4-mini (OpenAI 2025b), DeepSeek-R1 (Guo et al. 2025), and Qwen-3-32B (Yang et al. 2025), as well as prior open-source and proprietary spreadsheet agents across three public benchmarks. On SheetBench, SheetBrain demonstrates significantly superior performance, highlighting its capability to handle challenging sheets.
- We conduct extensive experiments and analyses, including module ablations that demonstrate the importance of both understanding and validation, evaluations of different tool roles highlighting the symbolic dataflow, and investigations into effective spreadsheet encoding strategies. These provide valuable insights and guidance for future spreadsheet agent designs. Additionally, we present case studies comparing existing proprietary agents with SheetBrain, revealing key findings.

SheetBrain

SheetBrain employs a robust three-stage pipeline: understand, execute, and validate. In the understanding stage, the LLM generates both a sheet summary and task-specific information to develop a global comprehension of the spreadsheet, laying the foundation for the subsequent stages. During execution, the LLM operates within code sandbox environment equipped with preloaded table-processing libraries and a customized Excel helper toolkit, enabling effective neuro-symbolic reasoning. In the validation stage, the LLM evaluates whether the execution reasoning and results are reasonable. If inconsistencies arise, the execution module incorporates feedback from the validation module to refine its reasoning and re-execute, iteratively enhancing the outcome.

Understanding Module

The understanding module serves as SheetBrain's cognitive front-end, transforming a user query and raw spreadsheet data into valuable prior knowledge. As shown in Figure 2, given the raw spreadsheet data and user query, we prompt the LLM to generate a sheet summary that includes the sheet's purpose, object relationships and structures, as well as problem insights indicating which parts of the spreadsheet data are relevant to the query. To balance providing sufficient context with computational constraints—especially for ultra-large spreadsheets—it employs dynamic token budget management, allocating a token budget per sheet to capture essential information while avoiding context window overflow. Within this budget, we utilize an enhanced markdown serialization that encodes cell content, cell position notation, and merged cell annotations. This approach preserves crucial 2D spatial context, enabling the agent to perceive the sheet's layout in a manner that mirrors human visual interpretation and reasoning. The detailed information generated by this module is listed as follows:

- **Sheet Summary:** An overview outlining the workbook's overall objective, key relationships (e.g., between sheets), and data structure organization, as well as conducting an in-depth structural analysis, for example, as shown in Figure 2, by recognizing that "each user's data occupies a distinct multi-row block", the summary guides the execution module to iterate over blocks rather than individual rows.
- **Problem-Specific Insights:** Insights derived directly from the user's query, which include identifying important rows and columns, recommending focus areas, and suggesting strategic approaches for the execution module. The recommendation to *"Filter users with chat service count* > 11*"* exemplifies this by providing a direct, executable strategy that prunes the search space, ensuring the final execution is both efficient and accurate.

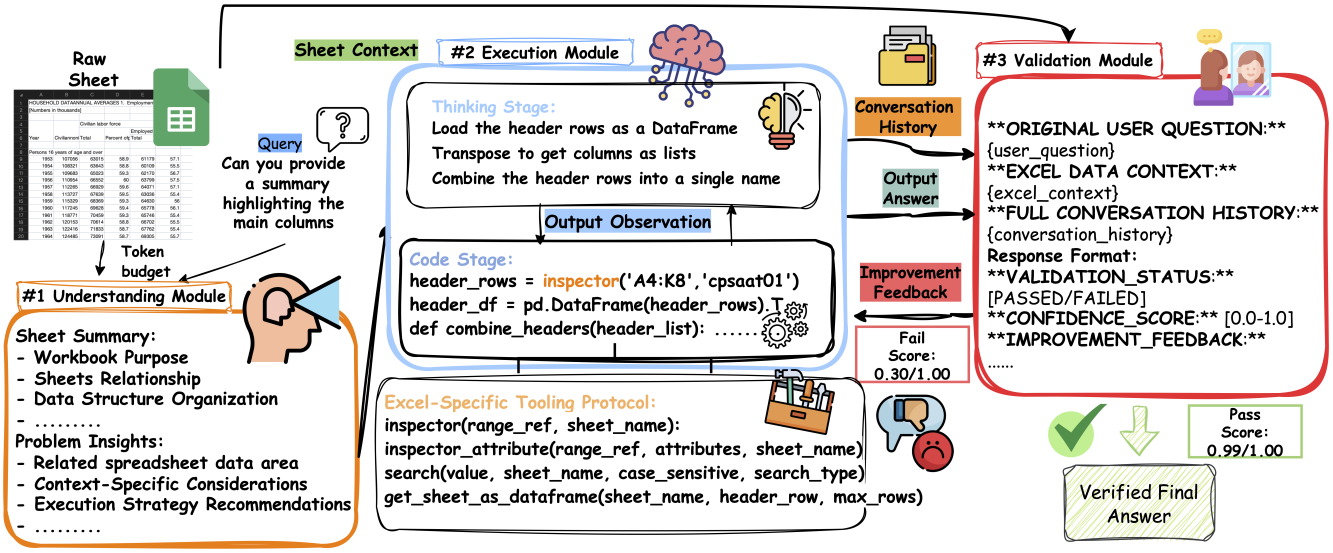

Figure 1: SheetBrain Pipeline. SheetBrain adopts a robust three-stage pipeline: understand, execute, and validate. It first constructs a global, query-aware understanding of the spreadsheet, then performs tool-augmented reasoning in a Python sandbox, and finally validates and refines its output through iterative self-correction.

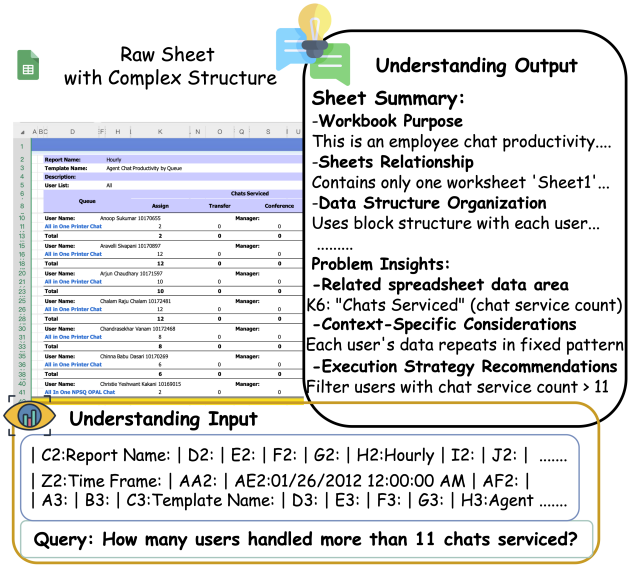

Figure 2: The workflow of the understanding module.

Execution Module

The execution module serves as the operational core of SheetBrain, where task decomposition, code generation, and data manipulation occur. The module is built upon three main components: an Excel-specific tooling protocol; a symbolic dataflow architecture equipped with a code sandbox; and an iterative reasoning and execution cycle. This design allows the agent to have several interactive turns to perform neuro-based reasoning or treat Excel files as manipulable variables, providing a persistent environment for symbolic computation.

Excel-Specific Tooling Protocol Our Excel-specific tooling protocol is empirically designed based on systematic analysis of common operational patterns and failure modes encountered by agents working with spreadsheets. The protocol encapsulates these insights into a set of specialized Python functions that integrate neural reasoning with symbolic execution. It enables seamless conversion of spreadsheet data into standard Python data structures, facilitating complex analyses through symbolic code. For example, the protocol incorporates robust search capabilities that support exact matches as well as partial and whitespace-tolerant queries, enhancing resilience to noisy or ambiguous inputs. Additionally, it provides fine-grained cell inspection, including extraction of visible values, formatting attributes (e.g., cell color, font), and underlying formulas. These features allow agents to leverage critical visual and structural cues essential for understanding spreadsheets in real-world scenarios.

Symbolic Dataflow Architecture Unlike conventional agents that rely on neural dataflow—embedding raw tool outputs back into the LLM's context—SheetBrain employs a symbolic dataflow architecture. This design handles queries by storing the sheet in a named variable, and generating code that directly references this variable to perform operations like counting the occurrences of values in a column, enabling accurate and precise answers (see Appendix Case 2). In contrast, neural dataflow approaches return thousands of raw data entries into the LLM's context, consuming excessive tokens that may exceed the context limit and hinder accurate computation. By offloading operations to the Python interpreter, SheetBrain ensures computational fidelity and scalable memory, allowing it to process large spreadsheets

where neural dataflow agents face fundamental limitations.

Iterative Reasoning and Execution Cycle The execution module does not follow a single-shot execution paradigm. Instead, it adopts an iterative cycle alternating between reasoning and execution. In each iteration, the agent performs reasoning based on the query, the current symbolic state, and prior outputs, then generates corresponding code. This code is executed within a sandbox, which updates the symbolic state with new or modified variables. The resulting outputs—whether values, transformed DataFrames or errors—are returned as feedback to the agent. This feedback guides the agent's subsequent reasoning, enabling it to refine its strategy, correct errors, or advance to the next logical subtask. For example, the agent may first load a sheet, then apply a filter, and subsequently aggregate the filtered data. This cycle continues until the agent determines that the task is complete and the current output satisfies the user query.

Validation Module

The validation module serves as the final and critical stage in the SheetBrain pipeline, functioning as a quality assurance and self-correction mechanism. It is invoked after the execution module generates a provisional answer, and is responsible for systematically evaluating the result before producing the final output. This evaluation is guided by a structured checklist and is informed by multiple inputs: the spreadsheet preview, the reasoning trace, the execution logs, the original user query, and the proposed answer. These inputs are cross-referenced to verify logical consistency, correctness, and relevance. The checklist assesses key dimensions such as data handling (e.g., extraction, transformation accuracy) and answer quality (e.g., completeness, format adherence, and alignment with the user query). The module outputs a binary decision—pass or fail—along with a confidence score ranging from 0 to 1. In case of failure, it provides detailed diagnostic feedback and concrete suggestions for improvement, which are forwarded to the execution module. This feedback initiates another execution cycle, enabling the agent to revise its approach and correct errors. The process repeats iteratively until the validation module returns a positive judgment. Only then is the final answer delivered to the user. This closed-loop feedback architecture is crucial for achieving high reliability and precision in complex spreadsheet reasoning tasks.

SheetBench

To address the gap in existing spreadsheet benchmarks, which focus on small, well-structured tables and overlook real-world complexity, we carefully developed SheetBench—a comprehensive benchmark designed to evaluate methods' capabilities in challenging spreadsheet QA and manipulation tasks. Compiled from 11 public sources—including HiTab (Cheng et al. 2021), MimoTable (Li et al. 2024b), RealHiTBench (Wu et al. 2025b), MultiHiertt (Zhao et al. 2022), DabStep (Egg et al. 2025), SheetCoPilot (Li et al. 2023), SpreadsheetBench (Ma et al. 2024), SheetAgent (Chen et al. 2024), NCSE (for Science and Statistics 2025), MMQA (Wu et al. 2025a), and CodaBench (Xu et al. 2022)—it contains 69 cases carefully selected by human annotators for their challenge to existing LLM agents such as ChatGPT (OpenAI 2023) and BizChat (Microsoft Corporation 2025). During the curation process, we also corrected queries and sheet issues that could cause misleading errors, ensuring these 69 cases maintain high quality and reliability. The benchmark systematically assesses methods across four key challenges—understanding complex structures, reasoning over multiple tables, scaling to large sheets, and performing spreadsheet editing. Covering a wide range of tasks from QA to manipulation involving formula propagation and layout modifications, SheetBench provides a rigorous and holistic testbed for advancing spreadsheet intelligence.

Experiments and Analysis

Experimental Setup

Benchmarks To comprehensively evaluate the performance of SheetBrain, we conducted extensive experiments across multiple benchmarks. Our evaluation covers three public datasets: MultiHiertt (Zhao et al. 2022) (for numerical reasoning over multi-hierarchical tabular and textual data), RealHiTBench (Wu et al. 2025b) (featuring complex, real-world hierarchical tables for LLM-based table analysis), and SpreadsheetBench (Ma et al. 2024) (a challenging benchmark for real-world spreadsheet manipulation tasks), all of which allowed us to evaluate capabilities in table-based reasoning and operations. Additionally, we introduce our own benchmark, SheetBench, specifically designed to address the challenges posed by large, multi-table, and structurally complex spreadsheets.

Baselines Our comparison targets include two categories of models: (1) vanilla LLMs, encompassing several open-source and closed-source models accessed via API, including GPT-4.1 (OpenAI 2025a), o4-mini (OpenAI 2025b), 4o (Hurst et al. 2024), Qwen-3-32b (Yang et al. 2025), and DeepSeek-R1 (Guo et al. 2025); and (2) specialized spreadsheet agents, namely StructGPT and SheetAgent. To ensure a fair comparison, all agent-based models, including our SheetBrain, utilize GPT-4.1 as their backbone LLM. In our implementation of SheetBrain, we set a 10,000-token budget for the initial data preview. The spreadsheet's structure is serialized using a Markdown format that includes A1-style cell position notation and explicit indicators for merged cells.

Evaluator For SpreadsheetBench, we use its official evaluators to assign scores. For evaluating MultiHierTT, RealHiTBench, and SheetBench, we employ an automated LLM-as-judge evaluator (Li et al. 2024a), co-developed alongside the SheetBench dataset, to ensure consistent and scalable correctness assessment. It is important to note that our judge does not "grade open-ended text." Instead, for QA tasks, it performs a **canonicalized comparison** between the model's answer and the gold reference from the benchmark. We found this approach to be significantly more robust than raw string exact match (EM) for spreadsheet answers, which often include variations in units, formatting, or minor numeric discrepancies. Our evaluator prompt is provided in the

Model	Multihiertt	Spreadsheetbench			Realhitbench				Sheetbench
		Cell	Sheet	**Overall**	Fact Checking	Data Analysis	Numerical Reasoning	**Overall**	
Vanilla Models									
gpt-4.1	53.5	NA	NA	NA	75.0	71.1	70.4	70.0	50.7
o4-mini	54.2	NA	NA	NA	76.7	71.4	77.0	71.7	52.0
4o	51.8	NA	NA	NA	74.3	63.9	71.4	67.7	50.7
Qwen3-32b	50.2	NA	NA	NA	73.9	71.4	66.7	68.1	50.7
Deepseek-R1	51.1	NA	NA	NA	69.6	50.0	66.7	64.6	52.0
Other Models									
StructGPT	13.0	NA	NA	NA	44.3	30.8	42.9	40.3	21.3
SheetAgent	35.3	14.3	33.7	21.8	70.1	31.8	68.6	58.8	50.8
SheetBrain (ours)	**62.6**	**35.4**	**37.8**	**36.4**	**85.5**	**71.8**	**73.6**	**78.3**	**80.3**

Table 2: Main Results: Performance of various models across four evaluation datasets. Each dataset may have multiple sub-tasks (e.g., fact-checking). The evaluation for sheetbench focuses solely on question-answering tasks, and the visualization category has been excluded from realhitbench. **NA** indicates the model does not support the required task capability

Appendix.

Main Results

As shown in Table 2, across all benchmarks, SheetBrain achieves substantial performance gains over existing methods. It attains state-of-the-art results on all QA and manipulation benchmarks, and also demonstrates superior performance on the more challenging cases in SheetBench.

QA Compared with vanilla LLMs, GPT-4.1-based SheetBrain improves accuracy by 9.1% over vanilla GPT-4.1, and also outperforms other LLMs and agents in MultiHiertt. In RealHitBench, SheetBrain improves overall performance by 8.3%, with a notable gain of over 10% on fact-checking, as well as improvements in data analysis and numerical reasoning, compared to vanilla GPT-4.1. This suggests that our approach is particularly effective in scenarios that demand precise cross-referencing and consistency checks, which may benefit from the inherent validation mechanisms in our design. The relatively smaller gains in data analysis and numerical reasoning imply that these tasks are more execution-driven and less dependent on iterative validation.

Manipulation Within the set of baselines, only SheetAgent possesses table-editing capabilities; accordingly, we conduct a direct comparison between SheetBrain and SheetAgent. Experimental results demonstrate that SheetBrain surpasses SheetAgent by nearly 15% in accuracy, with even greater gains observed at the cell level. These findings underscore the superior fine-grained capability of our approach in performing precise table manipulations.

Ablation Studies

We conducted a series of ablation studies to dissect the specific contributions of each key component within the SheetBrain framework.

Ablations on Understanding and Validation Modules. We assess the contributions of the understanding and validation modules. The results show that removing the understanding module leads to a 3.3% performance drop on both RealHitBench and SheetBench. Similarly, disabling the validation module results in a comparable performance decline, confirming its critical role in verifying and correcting the execution process. When both modules are removed simultaneously, performance further degrades to 73.3% on RealHitBench and 73.8% on SheetBench, which confirms that these components offer complementary advantages and jointly enhance the system's overall accuracy.

Ablations on Components of Execution Module. We analyze the internal components of the execution module in 4. With the full suite of custom sheet toolkit, our method achieves an accuracy of 79.1% on SheetBench. When the `inspector` tool is removed, accuracy drops to 77.3%, and it further decreases to 73.1% when the `search` tool or all tools are removed. More importantly, when we replaced our code sandbox with a traditional neural calling (JSON-based) approach, the accuracy plummeted to 65.1%. This stark difference powerfully demonstrates the overwhelming advantage of our code sandbox architecture and its integrated symbolic tool suite for handling data-intensive spreadsheet tasks.

Ablations on Serialization Strategies for Sheet Content. to investigate the impact of serialization strategies for sheet content on the performance of SheetBrain, we evaluate a variety of serialization formats, grouped into two main categories: Markdown-based and HTML-based representations. The Markdown-based approaches include a plain markdown format without positional metadata, as well as a variant that incorporates explicit cell position information. In contrast, the HTML-based methods include a plain HTML table representation, an HTML structure that mimics markdown style with added cell positions, a format that adds both cell positions and colspan attributes, and finally, a variant that utilizes row tags and colspan, while omitting explicit column positions. Table 5 presents the performance comparison across these serialization schemes. The results indicate that HTML-based serializations perform slightly better than their Markdown-based counterparts. More notably, the inclusion of cell position information yields a significant performance improvement across both categories, highlighting the importance of spatial context in structured data understanding. Interestingly, among the HTML variants, the format employing row tags with colspan, but without explicitly specifying

Understanding	Validation	Realhitbench	SheetBench
✓	✓	78.3	80.3
✗	✓	75.0	77.0
✓	✗	76.7	77.0
✗	✗	73.3	73.8

Table 3: Ablation study with different components removed. ✓ indicates the module is used; ✗ indicates it is removed. Results are shown as hit rate percentages.

Execution Method	Accuracy (%)
Code Sandbox Approach	
Full Method (All Tools)	79.1
w/o Inspector Tool	77.3
w/o Search Tool	73.1
w/o All Tools	73.1
Traditional Tool-Use Approach	
Neural Calling (JSON-based)	65.1

Table 4: Ablation study of the execution module's components on the SheetBench dataset. We compare our full method using a code sandbox against variants with specific tools removed, and against a traditional neural calling (JSON-based) approach. The results highlight the significant performance gains from our integrated tool suite and the code sandbox architecture for data-intensive tasks.

cell positions, outperforms the version that encodes precise cell positions. This suggests that providing high-level row structure and span information may be more beneficial than overly detailed positional encoding, possibly due to reduced noise or increased generalizability.

Qualitative Analysis and Insights

Strategy Adaptation Based on Table and Query Characteristics. To gain a deeper understanding of SheetBrain's performance advantages and internal mechanisms, we conduct a qualitative analysis, which reveals a complex trade-off between symbolic computation and neural reasoning.

- **Symbolic computation** excels in scenarios involving large or extra-large tables and tasks requiring complex, multi-step calculations. For example, when a task requires filtering a 100,000-row table and then computing a conditional average, the neural approach is infeasible. The entire dataset cannot be encoded into the context. SheetBrain's symbolic approach handles this gracefully by loading the data into a pandas dataframe and executing a few lines of code, demonstrating a significantly higher performance ceiling for data-intensive operations.
- **Neural reasoning** also shows competitive potential in specific niche cases: small-to-medium-sized tables with complex hierarchical structures. When the entire table can be fully encoded within the LLM's context window, the model can sometimes leverage its powerful pattern-matching capabilities to directly comprehend a complex layout (e.g., intricate multi-level headers) holistically. For instance, Case 4 perfectly illustrates this niche scenario: the symbolic approach failed on a simple query about fishery landings because it overlooked the crucial **"of which" breakdown items** within the complex row headers, leading to an incorrect parse. By contrast, neural reasoning, with the entire table encoded in its context, leveraged its holistic pattern-matching capabilities to directly comprehend this indented layout and accurately extract the answer. In such a scenario, a direct neural inference can be more straightforward than a step-by-step symbolic process of inspecting and parsing the structure through code.

Encoding Type	Variant	Accuracy
Markdown	Pure Markdown	63.3
	With Cell Position	75.0
HTML	Pure HTML	59.7
	MD-like HTML + Cell Pos.	76.3
	HTML + Colspan + Cell Pos.	76.0
	HTML + Colspan + Row Tag	76.7

Table 5: Accuracy of different table encoding variants.

The observation suggests that guiding the agent to differentiate its strategy based on the table characteristics and query type can lead to performance gains. Specifically, prompting the agent to output its thought process for small-to-medium tables with complex hierarchies, while defaulting to executing code for large tables or complex multi-step calculations, improves overall QA accuracy. This dynamic, scenario-based strategy selection allows the agent to better leverage the respective strengths of both neural reasoning and symbolic computation.

Limitations of Spreadsheet Previews in Existing Agents. Current agents like ChatGPT (OpenAI 2023) often struggle with spreadsheet analysis because they rely on a single, limited method for previewing data. By only loading and displaying the first few rows (e.g., using `df.head()`), they get a severely restricted view of the sheet's structure. This blind approach is a major limitation, especially when dealing with spreadsheets that have complex hierarchical layouts or multiple tables. As demonstrated in case analysis 1, this can lead to a fundamental misinterpretation of the data, such as failing to correctly identify a non-standard header row. This initial misunderstanding can cause the entire analysis to fail. This very observation, confirmed by poor performance on complex and multi-table categories in tests like SheetBench, highlights the critical need for a more comprehensive understanding module that can provide a full content overview before attempting any analysis.

The Need for Global Verification During Execution. Our error analysis revealed that even symbolic methods have specific failure patterns. During the code execution and output observation process, the agent can become overly focused on the local information returned from a single step, causing it to ignore the global context. For instance, as

Model	Complex	Multi	Large Sheet	Edit	Total
Test Cases	21	20	20	8	69
Vanilla Models					
4o	16	16	2	NA	34
o3	19	15	4	NA	38
o4-mini	16	16	3	NA	35
o3-mini	16	17	4	NA	37
4.1	16	17	1	NA	34
Qwen-3-32b	17	16	1	NA	34
Deepseek-R1	16	17	2	NA	35
BizChat Analyst	18	14	9	6	47
ChatGPT (4o)	13	8	9	NA	30
StructGPT (4.1)	12	1	0	NA	13
SheetAgent (4.1)	11	10	10	4	35
SheetBrain (4.1)	20	18	11	6	55

Table 6: Sheetbench Results.

shown in Case 3, when tasked with summing landings data, the agent correctly extracted numerical values from the sheet but failed to recognize the hierarchical structure, leading to a significant double-counting error by including both parent rows and their "of which" sub-categories in the total. This finding underscores the necessity of a global verification step and ultimately led us to incorporate a validation module. This module acts as a quality assurance and self-correction mechanism, prompting the agent to rigorously evaluate its entire process against the initial data structure before producing the final answer, thereby ensuring high reliability by catching and rectifying such logical flaws.

SheetBench Analysis

We evaluate a total of 12 models, grouped into vanilla LLMs and spreadsheet agents, on our proposed benchmark, SheetBench. The benchmark comprises 69 challenging tasks across four categories: Complex Tables, Multi-table Layouts, Large Sheets, and Editing Operations. Each model is evaluated on the same set of cases to ensure fairness and comparability. In addition, we provide a standardized prompting-based evaluator to assess QA outputs in a consistent and reproducible manner. The detailed performance breakdown is shown in Table 6. It demonstrates especially strong performance on the Complex and Multi-table subsets, achieving 20 out of 21 and 18 out of 20 respectively. These results highlight its superior capability in handling hierarchical structures, interleaved tables, and messy layouts that frequently appear in real-world spreadsheets.

Related Work

LLM Prompting and Fine-tuning for Tabular Data

LLMs have shown increasing promise for reasoning over structured tabular data, including question answering, semantic parsing, and data generation (Fang et al. 2024). Early work demonstrates that general-purpose LLMs such as GPT-3 (Brown et al. 2020) can answer table-based queries through in-context prompting (Chen 2022). To enhance compositional reasoning, DATER (Ye et al. 2023) decomposes both queries and tables into semantic units. TAPEX (Liu et al. 2021) advances table pretraining by fine-tuning a neural SQL executor on synthetic query–answer pairs, achieving state-of-the-art results on multiple benchmarks. Reasoning strategies have also evolved: chain-of-thought prompting (Wei et al. 2022) enables models to perform symbolic reasoning step by step, while self-consistency decoding (Wang et al. 2022) improves accuracy by aggregating diverse reasoning paths. Binder (Cheng et al. 2023) shows that Codex (Chen et al. 2021) can generate SQL programs from natural language, demonstrating strong semantic parsing capabilities. StructGPT (Jiang et al. 2023) builds on this by introducing a hierarchical reasoning framework tailored to structured tables. Despite these advances, most approaches are optimized for clean, single-table inputs and struggle with the complex, irregular, and layout-heavy structures common in real-world spreadsheets.

Spreadsheet Agents

Frameworks such as ReAct (Yao et al. 2023) have promoted recent efforts to move beyond static prompting and toward interactive spreadsheet agents (Krishnan 2025). SheetCopilot (Li et al. 2023) formulates spreadsheet interaction as a software agent task, using a finite state machine and atomic operations to support task execution across a large benchmark suite. SheetAgent (Chen et al. 2024) incorporates planning, summarization, and code retrieval mechanisms to enable long-horizon reasoning over spreadsheets. SheetMind (Zhu et al. 2025) extends this line with a multi-agent architecture that decomposes user commands, translates intentions into formal grammar-based plans, and performs actions in real-time using the Google Sheets API. These agents represent a shift toward autonomous spreadsheet reasoning, yet many of them still operate on simplified inputs or assume clean table boundaries. Challenges such as imprecise user queries and mixed data types remain significant obstacles for robust deployment in enterprise or messy real-world spreadsheets.

Conclusion

In this work, we introduce SheetBrain, a reasoning agent with symbolic dataflow based on a novel understand-execute-validate framework for reasoning over complex, large-scale spreadsheets. SheetBrain combines a deep understanding module, a symbolic execution engine operating within code sandbox, and a self-correcting module to address intricate spreadsheet tasks that exceed the capabilities of purely neural models. In addition, we present SheetBench, a new benchmark comprising realistic spreadsheet scenarios with multi-table layouts, hierarchical structures, and large data volumes. Experimental results demonstrate that SheetBrain consistently outperforms strong vanilla LLMs from OpenAI, Qwen, and DeepSeek, as well as existing spreadsheet agents, across both public datasets and SheetBench. Our findings highlight the importance of symbolic dataflow for scalable and precise computation, and show that structural understanding is essential for mitigating the limited visibility and context fragmentation that hinder current LLM-based spreadsheet reasoning approaches.

References

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. D. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Chen, W. 2022. Large language models are few (1)-shot table reasoners. *arXiv preprint arXiv:2210.06710*.

Chen, Y.; Yuan, Y.; Zhang, Z.; Zheng, Y.; Liu, J.; Ni, F.; and Hao, J. 2024. Sheetagent: A generalist agent for spreadsheet reasoning and manipulation via large language models. In *ICML 2024 Workshop on LLMs and Cognition*.

Cheng, Z.; Dong, H.; Wang, Z.; Jia, R.; Guo, J.; Gao, Y.; Han, S.; Lou, J.-G.; and Zhang, D. 2021. Hitab: A hierarchical table dataset for question answering and natural language generation. *arXiv preprint arXiv:2108.06712*.

Cheng, Z.; Xie, T.; Shi, P.; Li, C.; Nadkarni, R.; Hu, Y.; Xiong, C.; Radev, D.; Ostendorf, M.; Zettlemoyer, L.; Smith, N. A.; and Yu, T. 2023. Binding Language Models in Symbolic Languages. *ICLR*, abs/2210.02875.

Egg, A.; Goyanes, M. I.; Kingma, F.; Mora, A.; von Werra, L.; and Wolf, T. 2025. DABstep: Data Agent Benchmark for Multi-step Reasoning. *arXiv preprint arXiv:2506.23719*.

Fang, X.; Xu, W.; Tan, F. A.; Zhang, J.; Hu, Z.; Qi, Y.; Nickleach, S.; Socolinsky, D.; Sengamedu, S.; and Faloutsos, C. 2024. Large Language Models (LLMs) on Tabular Data: Prediction, Generation, and Understanding–A Survey. *arXiv preprint arXiv:2402.17944*.

for Science, N. C.; and Statistics, E. 2025. NCSES - National Center for Science and Engineering Statistics. Accessed: 2025-07-26.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Hurst, A.; Lerer, A.; Goucher, A. P.; Perelman, A.; Ramesh, A.; Clark, A.; Ostrow, A.; Welihinda, A.; Hayes, A.; Radford, A.; et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Jiang, J.; Zhou, K.; Dong, Z.; Ye, K.; Zhao, W. X.; and Wen, J.-R. 2023. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Krishnan, N. 2025. Ai agents: Evolution, architecture, and real-world applications. *arXiv preprint arXiv:2503.12687*.

Li, H.; Dong, Q.; Chen, J.; Su, H.; Zhou, Y.; Ai, Q.; Ye, Z.; and Liu, Y. 2024a. Llms-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*.

Li, H.; Su, J.; Chen, Y.; Li, Q.; and Zhang, Z.-X. 2023. Sheetcopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems*, 36: 4952–4984.

Li, Z.; Du, Y.; Zheng, M.; and Song, M. 2024b. MiMoTable: A Multi-scale Spreadsheet Benchmark with Meta Operations for Table Reasoning. *arXiv preprint arXiv:2412.11711*.

Liu, Q.; Chen, B.; Guo, J.; Ziyadi, M.; Lin, Z.; Chen, W.; and Lou, J.-G. 2021. TAPEX: Table pre-training via learning a neural SQL executor. *arXiv preprint arXiv:2107.07653*.

Ma, Z.; Zhang, B.; Zhang, J.; Yu, J.; Zhang, X.; Zhang, X.; Luo, S.; Wang, X.; and Tang, J. 2024. SpreadsheetBench: Towards Challenging Real World Spreadsheet Manipulation. arXiv:2406.14991.

Microsoft Corporation. 2025. Microsoft 365 Copilot. https://m365.cloud.microsoft/. Accessed on 2025-08-02.

OpenAI. 2023. ChatGPT: Optimizing Language Models for Dialogue. https://openai.com/chatgpt. Accessed on 2025-08-02.

OpenAI. 2025a. Introducing GPT-4.1 in the API. Accessed: 2025-08-01.

OpenAI. 2025b. OpenAI o3 and o4-mini System Card. Accessed: 2025-08-01.

Pan, Z.; Luo, H.; Li, M.; and Liu, H. 2024. Chain-of-action: Faithful and multimodal question answering through large language models. *arXiv preprint arXiv:2403.17359*.

Wang, C.; Deng, Y.; Lyu, Z.; Zeng, L.; He, J.; Yan, S.; and An, B. 2024. Q*: Improving multi-step reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; and Zhou, D. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Wu, J.; Yang, L.; Li, D.; Ji, Y.; Okumura, M.; and Zhang, Y. 2025a. MMQA: Evaluating LLMs with Multi-Table Multi-Hop Complex Questions. In *The Thirteenth International Conference on Learning Representations*.

Wu, P.; Yang, Y.; Zhu, G.; Ye, C.; Gu, H.; Lu, X.; Xiao, R.; Bao, B.; He, Y.; Zha, L.; et al. 2025b. RealHiTBench: A Comprehensive Realistic Hierarchical Table Benchmark for Evaluating LLM-Based Table Analysis. *arXiv preprint arXiv:2506.13405*.

Xu, S.; Xie, W.; Zhao, L.; and He, P. 2025. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*.

Xu, Z.; Escalera, S.; Guyon, I.; Pavao, A.; Richard, M.; Tu, W.-W.; Yao, Q.; and Zhao, H. 2022. Codabench: Flexible, Easy-to-Use and Reproducible Benchmarking Platform.

Yang, A.; Li, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Gao, C.; Huang, C.; Lv, C.; et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Ye, Y.; Hui, B.; Yang, M.; Li, B.; Huang, F.; and Li, Y. 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.

Zhao, Y.; Li, Y.; Li, C.; and Zhang, R. 2022. MultiHiertt: Numerical reasoning over multi hierarchical tabular and textual data. *arXiv preprint arXiv:2206.01347*.

Zhu, R.; Cheng, X.; Liu, K.; Zhu, B.; Jin, D.; Parihar, N.; Xu, Z.; and Gao, O. 2025. SheetMind: An End-to-End LLM-Powered Multi-Agent Framework for Spreadsheet Automation. *arXiv preprint arXiv:2506.12339*.