Towards Robust Real-World Spreadsheet Understanding with Multi-Agent Multi-Format Reasoning

Houxing Ren[1] **Mingjie Zhan**[2*] **Zimu Lu**[1] **Ke Wang**[1] **Yunqiao Yang**[1]
Haotian Hou[2] **Hongsheng Li**[1,3,4*]
[1]CUHK MMLab [2]SenseTime Research [3]Shenzhen Loop Area Institute [4]CPII under InnoHK
renhouxing@gmail.com zhanmingjie@sensetime.com hsli@ee.cuhk.edu.hk

Abstract

Spreadsheets are central to real-world applications such as enterprise reporting, auditing, and scientific data management. Despite their ubiquity, existing large language model based approaches typically treat tables as plain text, overlooking critical layout cues and visual semantics. Moreover, real-world spreadsheets are often massive in scale, exceeding the input length that LLMs can efficiently process. To address these challenges, we propose SpreadsheetAgent, a two-stage multi-agent framework for spreadsheet understanding that adopts a step-by-step reading and reasoning paradigm. Instead of loading the entire spreadsheet at once, SpreadsheetAgent incrementally interprets localized regions through multiple modalities, including code execution results, images, and LaTeX tables. The method first constructs a structural sketch and row/column summaries, and then performs task-driven reasoning over this intermediate representation in the Solving Stage. To further enhance reliability, we design a verification module that validates extracted structures via targeted inspections, reducing error propagation and ensuring trustworthy inputs for downstream reasoning. Extensive experiments on two spreadsheet datasets demonstrate the effectiveness of our approach. With GPT-OSS-120B, SpreadsheetAgent achieves 38.16% on Spreadsheet Bench, outperforming the ChatGPT Agent baseline (35.27%) by 2.89 absolute points. These results highlight the potential of SpreadsheetAgent to advance robust and scalable spreadsheet understanding in real-world applications. Code is available at https://github.com/renhouxing/SpreadsheetAgent.

1 Introduction

Spreadsheets are among the most widely used data formats in real-world applications, ranging from enterprise reporting and financial auditing to scientific data management and governmental records. Their structured nature makes them essential for storing, organizing, and analyzing large volumes of information. With the rapid development of large language models (LLMs) (OpenAI, 2023; Touvron et al., 2023; Jiang et al., 2024; Yang et al., 2024), table processing has emerged as an active research direction. A series of new systems, such as TableGPT (Li et al., 2024; Su et al., 2024), Chain-of-Table (Wang et al., 2024), TaPERA (Zhao et al., 2024b), SheetAgent (Chen et al., 2025), and SheetMind (Zhu et al., 2025), have been proposed to improve tasks like table understanding, reasoning, and interaction. These models highlight the potential of LLMs to enhance efficiency in practical scenarios, including automated reporting, statistical analysis, and intuitive spreadsheet manipulation.

However, despite these advances, most existing approaches represent tables as pure text, *e.g.,* Markdown (Zhao et al., 2024b; Li et al., 2024), HTML (Sui et al., 2024; Wu et al., 2025a), or LaTeX (Sui et al., 2024; Wu et al., 2025a), with only a few exploring image-based inputs (Zheng et al., 2024; Zhou et al., 2025). Real-world spreadsheets, by contrast, are far more complex: they contain hierarchical headers, multiple sheets, and rich visual cues such as font colors, shaded cells, and borders. These stylistic elements carry semantic information that text-only formats cannot fully capture. Moreover, practical spreadsheets often span thousands of rows and columns, exceeding the context length that current LLMs can efficiently handle. As a result, existing methods struggle to represent both the structural complexity and the scale of spreadsheets, limiting their effectiveness in downstream tasks such as question answering and knowledge extraction.

To overcome these challenges, we propose SpreadsheetAgent, a two-stage multi-agent framework built on step-by-step reading and reasoning,

*Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1906–1933
July 2-7, 2026

where an extraction agent collaborates with a vision range agent and a LaTeX range agent to process spreadsheets incrementally. Rather than ingesting the entire spreadsheet in one pass, our framework allows agents to iteratively feed the model with small regions in multiple formats, *e.g.,* code execution results, images, and LaTeX tables. In this way, the model reads and reasons incrementally under a tight context budget. The two stages operationalize this paradigm: the Structure Extraction Stage constructs a structural sketch and row/column summaries through iterative region inputs; the Solving Stage performs task-driven reasoning by consulting and, when necessary, extending this sketch. Compared with previous approaches, SpreadsheetAgent preserves layout semantics while avoiding the need to load the entire table at once.

More concretely, in the Structure Extraction Stage, the extraction agent drives the exploration: it scans the spreadsheet, identifies structural cues such as hierarchical headers, merged cells, and multi-sheet organization, and assembles concise row-level and column-level summaries. To support this process, the extraction agent issues specialized requests to two assistants who answer queries and verify representations while delegating conversions to auxiliary tools. The vision range agent formulates visual queries over selected ranges, invokes the image-conversion module to render snapshots when needed, interprets visual cues, and performs vision-based verification by cross-checking the evolving sketch against the rendered views. The LaTeX range agent formulates structural queries over ranges, invokes the LaTeX-conversion module to obtain symbolic tables that preserve hierarchical headers and alignment, interprets the resulting structures, and conducts LaTeX-based verification for faithfulness. Through this division of labor, the agents jointly construct a compact intermediate representation that faithfully captures both spreadsheet content and layout, providing a reliable foundation for downstream reasoning.

Our main contributions are as follows:

- We propose SpreadsheetAgent, a novel two-stage multi-agent framework for spreadsheet understanding, which adopts a step-by-step reading and reasoning paradigm, enabling progressive interpretation without exceeding the context-length limitations of LLMs.
- We design a structural exploration and reasoning pipeline that processes spreadsheets through localized region-based inputs in multiple formats, including code execution results, image snippets, and LaTeX expressions, thereby capturing a wide range of semantics.
- Extensive experiments demonstrate the effectiveness of SpreadsheetAgent. On SpreadsheetBench, our method achieves 38.16% with GPT-OSS-120B, substantially outperforming the ChatGPT Agent[1] (35.27%). This 2.89 absolute improvement highlights the capability of SpreadsheetAgent in handling complex spreadsheet reasoning tasks.

2 Related Work

2.1 Table Understanding

A large body of research has sought to improve how language models understand and reason over tables, often by designing specialized representations or reasoning procedures. Chain-of-Table (Wang et al., 2024) introduced step-by-step tabular reasoning by iteratively generating operations that transform tables into a chain of intermediate representations, enabling systematic decomposition of complex queries. SheetMind (Zhu et al., 2025) proposed an LLM-powered multi-agent framework for spreadsheet automation via natural language, which uses a Manager Agent, an Action Agent, and a Reflection Agent integrated with Google Sheets. Similarly, SheetAgent (Chen et al., 2025) proposed an LLM-based agent with Planner, Informer, and Retriever modules for enhanced spreadsheet reasoning and manipulation. ReAcTable (Zhang et al., 2024) extended this idea with external tool augmentation and a voting mechanism, improving robustness in table question answering. TaPERA (Zhao et al., 2024b) tackled compositional queries by decomposing them into sub-questions, executing Python programs to retrieve facts from tables, and synthesizing long-form answers from program outputs. Beyond symbolic reasoning, multimodal methods have also emerged. TabPedia (Zhao et al., 2024a) unified multiple visual table understanding tasks via concept synergy, leveraging dual encoders for high- and low-resolution image features. Similarly, RealHiT (Wu et al., 2025a) focused on hierarchical headers, prompting the model to first generate a header tree before performing reasoning. More recently, general frameworks have been proposed to address diverse challenges in table reason-

[1] https://openai.com/index/introducing-chatgpt-agent/

ing. TableMaster (Cao, 2025) introduced a unified system incorporating table-of-focus, verbalization, program-aided reasoning, and adaptive strategies to handle different table tasks. ROT (Zhang et al., 2025) proposed a row-by-row traversal strategy with post-traversal reflection, aiming to reduce hallucinations while maintaining efficiency.

2.2 Fine-tuning for Table Understanding

Another major research direction is to adapt pretrained models to table tasks via fine-tuning or instruction tuning. TableGPT (Li et al., 2024) pioneered this by synthesizing diverse table-related tasks from real-world tables and applying instruction tuning, yielding improved generalization. This inspired follow-up work such as TableGPT2 (Su et al., 2024), Table-LLaVA (Zheng et al., 2024), and SynTab (Zhou et al., 2025), which extended the focus to multimodal table understanding through the construction of large-scale instruction-tuning datasets. Motivated by the strong performance of DeepSeek-R1 (DeepSeek-AI et al., 2025), reinforcement learning (RL) has also become a central paradigm for enhancing reasoning in language models, leading to a surge of RL-based approaches. In the table domain, two representative lines of work have recently been proposed. Jin et al. (2025) focuses on small language models, introducing layout-aware pretraining and a mix-paradigm GRPO variant to improve robustness and consistency, achieving substantial gains over base models. In contrast, Wu et al. (2025b) combined region-guided reasoning with table-aware GRPO to enhance efficiency and accuracy, even surpassing much larger baselines. These studies highlight the promise of RL in advancing language-based tabular reasoning.

3 Methodology

In this section, we present our framework for spreadsheet understanding, shown in Figure 1. We begin by describing how to extract structural and semantic information from spreadsheets through a dedicated extraction module. Next, we present how to verify the extracted results against the original table using a verification module to enhance reliability. Finally, we provide an overview of the entire pipeline.

3.1 Extraction Module

The extraction module targets the full diversity of real-world spreadsheets rather than treating tables as flat matrices. In practice, spreadsheets frequently include (1) hierarchical headers that encode layered categorical semantics, (2) complex row/column semantics such as group totals, stubs, and sectioned indices. These layout signals determine how cell values should be interpreted and aggregated; ignoring them leads to misaligned schemas and brittle reasoning. Therefore, our design instructs the model to explicitly detect and preserve these structures during the extraction process.

Specifically, we adopt a prompt-driven pipeline that guides the LLM to identify structural cues and to summarize them alongside localized content. Concretely, the prompt enumerates the expected phenomena (hierarchical headers, merged cells, sheet-level sectioning, formatting-based semantics such as font color or shading) and asks the model to return a normalized schema that binds layout to semantics. This step converts loosely formatted spreadsheets into a faithful intermediate description while remaining robust to large tables via region-wise processing. To further ensure reliability for subsequent modules, the model is required to emit outputs in a YAML format. We choose this format because it is human-readable, captures nested structure with minimal loss, and is straightforward to parse programmatically. We empirically observe that the output format materially affects downstream performance: structured YAML reduces ambiguity, stabilizes parsing, and improves compatibility with task-specific reasoners compared to free-form text.

In addition to structured YAML outputs, which improve downstream interpretability and stability, we further equip the system with three auxiliary tools that enhance the model's reasoning over challenging spreadsheet phenomena. These tools complement the schema-driven approach by providing precise access to raw values, visual cues, and structural signals that are otherwise difficult to capture through text prompting alone.

- **Code Execution.** The system can directly execute Python code written by the model, enabling precise parsing of raw values, numerical computation, or programmatic validation. Unlike conversion-based tools, this mechanism provides exact results for arithmetic operations and custom logic that go beyond approximate text reasoning.
- **Vision Range Description Agent.** Given a selected range, the system converts it into an

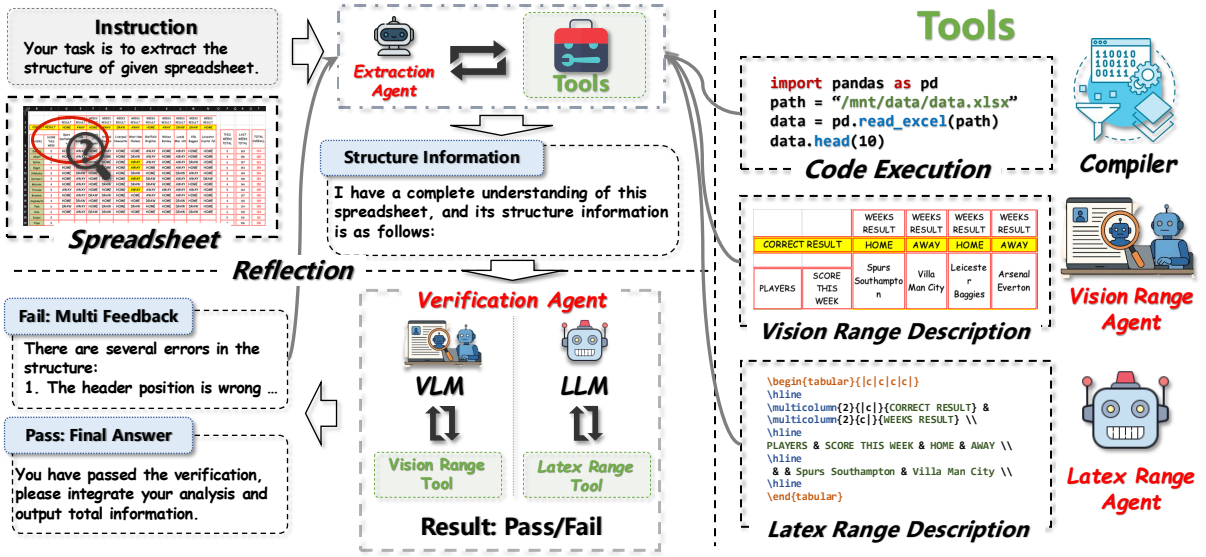

Figure 1: Overview of the proposed multi-agent framework for spreadsheet understanding. An Extraction Agent incrementally parses the spreadsheet with the help of code execution, vision range description, and LaTeX range description tools, while a Verification Agent cross-checks results using VLM-based and LLM-based range descriptions. Through iterative reflection and feedback, the system produces a faithful structural representation that preserves layout semantics under tight context budgets.

image, and then a vision-language model is prompted with the image and a question to extract semantic information from visual cues such as colors, borders.

- **LaTeX Range Description Agent.** Given a selected range, the system converts it into a LaTeX table through code, and then a language model is prompted with the LaTeX representation and a question to recover structural information such as hierarchical headers or merged cells.

By combining structure-aware prompting, a YAML-based intermediate, and tool-assisted mechanisms (code execution, vision, and LaTeX), the extraction module produces a compact representation that retains both content fidelity and layout semantics. This representation serves as the foundation for subsequent verification and task-driven reasoning.

3.2 Verification Module

For many tasks, verification is often easier than direct problem solving. Instead of asking the model to generate a complete interpretation from scratch, it is typically more reliable to let the model check whether an existing extraction is correct. This motivates our design of a verification module, which improves accuracy by validating the outputs of the extraction module before they are consumed in downstream reasoning.

Specifically, given the entire spreadsheet and its extracted YAML representation, the verification agent checks whether the extraction faithfully reflects the original table. Instead of reprocessing everything at once, which would exceed context limits for large spreadsheets, the agent selectively focuses on uncertain or structurally complex regions. It actively decides which regions to convert into alternative formats (image or LaTeX) for further inspection, thereby reducing error propagation and ensuring more reliable inputs for downstream reasoning.

- **Vision Verification Agent.** The agent may convert a selected region into an image and query a vision-language model to check whether the extracted schema matches the visual layout. Detected issues—such as missing merged cells, misinterpreted colors, or overlooked chart elements—are reported with suggested corrections.
- **LaTeX Verification Agent.** The agent may alternatively render a region as a LaTeX table and prompt a language model to verify structural fidelity. Features like hierarchical headers, multicolumn or multirow spans, and alignment are cross-checked against the YAML extraction, with any inconsistencies flagged and accompanied by revision proposals.

In this design, the verification module does not attempt to reprocess the entire table at once. Instead, it strategically combines the extracted YAML with targeted tool-assisted inspections. By leveraging adaptive region selection and focusing on checking rather than solving, the framework increases robustness and ensures that downstream reasoning is grounded in faithful representations of spreadsheet content and structure.

3.3 Discussion

As described in Algorithm 1, the two modules described above operate in a tightly-coupled, iterative loop rather than as independent components. The extraction module first produces a structured intermediate representation that preserves both spreadsheet content and layout. The verification module then inspects this representation against the original table, identifies potential errors, and proposes corrections. These corrections are fed back into the extraction process, which updates the representation accordingly. The cycle of extraction–verification–refinement continues until the schema achieves sufficient fidelity.

By combining extraction and verification in this iterative manner, our framework establishes a progressive reading-and-checking paradigm. Instead of attempting to process an entire spreadsheet in one step, the system incrementally builds and validates its understanding. This design offers two major benefits: (i) improved scalability, since only localized regions are processed at each step under a limited context budget; and (ii) enhanced accuracy, as verification reduces error propagation and strengthens reliability. Finally, once the representation has been validated, the resulting structured information is directly injected into the context of downstream tasks, ensuring that subsequent reasoning operates on a faithful and semantically enriched view of the spreadsheet.

4 Experiments

In this section, we present extensive experiments to demonstrate the effectiveness of the proposed method and analyze its performance. Due to the limited space, more detailed experiments are presented in Appendix A.

4.1 Experimental Setup

Test Dataset. We evaluate our method on SpreadsheetBench (Ma et al., 2024), a benchmark constructed from 912 real-world questions collected from Excel user forums. Each question is paired with its original spreadsheet file, which often contains multiple tables, non-standard relational structures, merged cells, and other non-textual elements such as pivot tables, charts, and annotations. In contrast, many existing benchmarks are constructed with highly regularized tables, such as WikiTQ (Pasupat and Liang, 2015) and TabFact (Chen et al., 2020), which cannot fully capture the irregular structures and noisy layouts often encountered in practical spreadsheet scenarios. Other benchmarks focus on hierarchical headers, such as HiTab (Cheng et al., 2022) and AIT-QA (Katsis et al., 2022), but represent them only in a serialized textual form (*e.g.,* JSON), which strips away the layout and formatting cues that are essential for spreadsheet manipulation.

To further evaluate our method under settings that specifically emphasize hierarchical headers, we also conduct experiments on RealHiTBench (Wu et al., 2025a) in Appendix A.1, which provides Excel files with complex hierarchical headers. This allows us to isolate and analyze the effectiveness of our approach on tasks dominated by header complexity, complementing the broader coverage of SpreadsheetBench.

Implementation Details. In our experiments, we use GLM-4.5V[2] (Hong et al., 2025) as the vision–language model, serving both as the vision conversion tool and the vision verification model, and Qwen3-Coder-480B[3] (Yang et al., 2025) as the language model, which acts as the main model, the LaTeX conversion tool, and the LaTeX verification model. For inference acceleration, we adopt vLLM (Joshi et al., 2025), using greedy decoding with temperature set to 0 and top-p set to 1. Spreadsheet content is serialized into structured text and appended to the prompt. All models are evaluated with the same prompt template to ensure fairness, using a maximum context length of 4K tokens per round and allowing at most 20 rounds of tool calls in total. We benchmark all models under identical conditions. For the Spreadsheet Bench, we report accuracy as the evaluation metric, while for the RealHiT Bench, we adopt Exact Match (EM) and F1 scores. All experiments are conducted on 8 NVIDIA H800 GPUs with vLLM, leveraging tensor parallelism and expert parallelism for efficient

[2] https://huggingface.co/zai-org/GLM-4.5V

[3] https://huggingface.co/Qwen/Qwen3-Coder-480B-A35B-Instruct

Algorithm 1: Incremental Extraction-Verification Loop

Input: Spreadsheet S
Output: Verified structural representation $Y^\star$

$\mathcal{C} \leftarrow \{S,\ \text{Prompt},\ \text{Tool interfaces}\}, Y \leftarrow \emptyset, Y^\star \leftarrow \emptyset$;
repeat
  while *the agent invokes a tool* **do**
    Select tool $T \in \{\texttt{Code}, \texttt{Vision}, \texttt{LaTeX}\}$ and parameters P;
    $O \leftarrow T(S, P)$;
    Append O to $\mathcal{C}$;
  end
  $Y \leftarrow \mathcal{C}[-1]$, *vision_pass* $\leftarrow$ false, *latex_pass* $\leftarrow$ false;
  $\mathcal{C}_v \leftarrow \{S,\ Y,\ \text{Verify prompt},\ \text{Vision interfaces}\}$;
  while *the vision agent invokes a vision tool* **do**
    Select parameters P_v;
    $R_v \leftarrow \texttt{VisionTool}(S, P_v)$;
    Append R_v to $\mathcal{C}_v$;
  end
  vision_pass, $\Delta_v \leftarrow \texttt{parse\_verification}(\mathcal{C}_v[-1])$;
  $\mathcal{C}_l \leftarrow \{S,\ Y,\ \text{Verify prompt},\ \text{LaTeX interfaces}\}$;
  while *the LaTeX agent invokes a LaTeX tool* **do**
    Select parameters P_l;
    $R_l \leftarrow \texttt{LaTeXTool}(S, P_l)$;
    Append R_l to $\mathcal{C}_l$;
  end
  latex_pass, $\Delta_l \leftarrow \texttt{parse\_verification}(\mathcal{C}_l[-1])$;
  if *vision_pass and latex_pass* **then**
    $Y^\star \leftarrow Y$;
    break;
  else
    Append Δ_v and Δ_l to $\mathcal{C}$;
  end
until *verification succeeds or max-iterations reached*;
return $Y^\star$

inference.

4.2 Evaluation

Baseline. We compare our method against a wide range of baselines, including both general-purpose LLMs (*e.g.,* GPT-3.5, GPT-4o, Llama-3) and spreadsheet-specific systems such as Sheet-Copilot (Li et al., 2023), Copilot in Excel[4], and ChatGPT Agent[5]. For fairness, the results of these methods are directly taken from the original SpreadsheetBench paper. In addition, we include Tree-Thinker (Wu et al., 2025a) as a strong recent approach for reasoning over hierarchical table structures. Human performance is further reported as an upper-bound reference.

Result. Table 1 reports the comparison results on SpreadsheetBench, from which we derive several key observations: (1) All existing systems still fall far short of human performance. Even GPT-4o, one of the most capable proprietary models, only achieves 18.35% under the soft restriction and 15.02% under the hard restriction, highlighting the inherent difficulty of real-world spreadsheet reasoning. (2) Our method establishes new state-of-the-art results. While the ChatGPT Agent was previously the strongest reported system with 35.27% accuracy under the soft setting, our framework achieves significant improvements. In partic-

[4] https://www.microsoft.com/microsoft-copilot
[5] https://openai.com/index/introducing-chatgpt-agent/

Table 1: Main experimental results on SpreadsheetBench. We report accuracy under both Soft Restriction and Hard Restriction settings, further broken down into cell-level, sheet-level, and overall scores. Numbers for existing methods are taken from the SpreadsheetBench paper.

Model	Soft Restriction (↑)			Hard Restriction (↑)		
	Cell-Level	Sheet-Level	Overall	Cell-Level	Sheet-Level	Overall
Binder (GPT-3.5)	1.58	0.05	1.17	0.00	0.00	0.00
Mixtral-8x7B	3.39	4.67	3.88	2.32	3.71	2.85
Llama-3-70B	1.13	7.90	3.74	0.71	7.14	3.18
GPT-3.5	3.33	13.11	7.09	2.50	9.97	5.37
GPT-4o	13.49	22.51	16.96	10.52	17.66	13.27
SheetCopilot	16.67	10.00	14.00	-	-	-
Copilot in Excel	23.33	15.00	20.00	-	-	-
ChatGPT Agent	38.27	30.48	35.27	-	-	-
GPT-OSS-20B	25.55	24.79	25.26	19.43	21.37	20.18
w/ TreeThinker	23.95	26.12	24.78	18.54	22.51	20.07
w/ SpreadsheetAgent	**34.46**	**28.49**	**32.16**	**27.81**	**25.64**	**29.25**
GPT-OSS-120B	30.78	27.64	29.57	24.96	23.93	24.56
w/ TreeThinker	32.14	29.82	31.25	24.42	26.50	25.22
w/ SpreadsheetAgent	**41.30**	**33.14**	**38.16**	**32.80**	**29.34**	**31.47**
Qwen3-30B	13.61	20.89	16.41	9.45	18.23	12.83
w/ TreeThinker	14.38	22.22	17.40	10.16	19.09	13.60
w/ SpreadsheetAgent	**20.62**	**25.17**	**22.37**	**16.04**	**22.22**	**18.42**
Qwen3-235B	20.50	25.83	22.55	15.33	22.51	18.09
w/ TreeThinker	22.99	26.50	24.34	17.29	23.65	19.74
w/ SpreadsheetAgent	**33.10**	**28.77**	**31.43**	**25.31**	**25.07**	**25.22**
Qwen3-Coder-480B	30.36	31.05	30.63	22.82	27.07	24.45
w/ TreeThinker	38.86	32.00	36.22	31.91	27.92	30.37
w/ SpreadsheetAgent	**45.63**	**35.33**	**41.67**	**36.90**	**31.05**	**34.65**
Human Performance	75.56	65.00	71.33	66.67	55.00	62.00

ular, SpreadsheetAgent with Qwen3-480B-A35B reaches 41.67% and with GPT-OSS-120B reaches 38.16%, surpassing the ChatGPT Agent across different model scales and demonstrating the robustness of our design. (3) Our framework consistently outperforms TreeThinker. A key limitation of TreeThinker is its exclusive focus on hierarchical headers, whereas our approach incorporates a broader spectrum of spreadsheet cues. By integrating visual signals and LaTeX-based representations, SpreadsheetAgent preserves richer layout and semantic information, leading to stronger generalization across the diverse structures present in SpreadsheetBench. (4) Beyond absolute accuracy, the results reveal consistent gains across different model scales. On smaller backbones such as Qwen3-30B, SpreadsheetAgent improves the overall score by nearly 6 points, while on large-scale models like Qwen3-Coder-480B, the gain exceeds 11 points. This trend confirms that the proposed framework is not restricted to high-capacity models but provides robust benefits across scales.

4.3 Detailed Analysis

4.3.1 Ablation Study

Here, we check how each component contributes to the final performance. We design several variants to examine the role of different components. Specifically, w/o Tools & Verify disables all external tools and the verification mechanism, and its sub-variants are: w/ JSON, which replaces YAML

Table 2: Ablation study of SpreadsheetAgent using Qwen3-30B as the reasoning model. We examine the contributions of verification, structural sketching, and multi-format tools.

Model	Soft Restriction (↑)			Hard Restriction (↑)		
	Cell-Level	Sheet-Level	Overall	Cell-Level	Sheet-Level	Overall
SpreadsheetAgent	20.62	**25.17**	**22.37**	**16.04**	**22.22**	**18.42**
w/o Tools & Verify	19.61	21.08	20.18	14.44	18.52	16.01
w/ JSON	19.37	22.98	20.76	13.90	20.23	16.34
w/o Structure	18.84	21.08	19.70	14.08	17.66	15.46
w/o Verify	20.44	23.08	21.45	15.69	20.51	17.54
w/o Vision Tool	**21.03**	22.13	21.45	15.33	19.37	16.89
w/o Latex Tool	19.01	22.41	20.32	14.44	19.09	16.23
w/o All	13.61	20.89	16.41	9.45	18.23	12.83

structure with JSON as the table representation; and w/o Structure, which does not have any format restrictions. In addition, w/o Verify disables the verification process, with two sub-variants: w/o Vision Tool, which removes the image conversion module; and w/o Latex Tool, which removes the LaTeX conversion module. Finally, w/o All eliminates all proposed components, leaving only the base reasoning agent.

Table 2 presents the ablation results of SpreadsheetAgent on SpreadsheetBench. Based on the results, several important findings can be drawn. First, the full model achieves the highest overall accuracy (22.37% / 18.42%), indicating that the joint use of all modules yields the strongest performance. Second, verification plays a critical role: removing it (w/o Verify) consistently lowers accuracy, while the even larger gap observed in w/o Tools & Verify highlights that tools and verification are mutually reinforcing rather than individually sufficient. Third, structural sketching proves essential. In the default setting, w/o Tools & Verify produces YAML outputs. We further evaluate w/ JSON and w/o Structure, which either replace YAML with JSON or discard sketching entirely. The results show that both YAML and JSON formats support relatively strong performance, as explicit structural representations allow downstream models to interpret the data more effectively. In contrast, removing structured outputs severely degrades accuracy, confirming that structural sketching is a key component for spreadsheet understanding. Fourth, the multi-format tools contribute in complementary ways. Removing the vision tool leads to slight gains in soft cell-level tasks but significantly harms performance under hard settings. Conversely, removing the LaTeX tool consistently reduces accuracy, underscoring its importance for preserving layout and structural fidelity. Finally, the w/o All variant results in a dramatic collapse (16.41% / 12.83%), nearly 10 points below the full model, which provides strong evidence that integrating all components into a unified framework is indispensable.

4.3.2 Effect of Structure Extractor

To further understand the impact of the structure extractor on the final performance, we conduct an ablation study where the extractor model is replaced while keeping the reasoning model fixed. Specifically, we substitute the Qwen3-480B extractor with two smaller candidates: Qwen3-30B and GPT-OSS-20B. Note that we do not replace GLM-4.5V, because the availability of LLMs that can both handle image inputs and support function calling is still very limited, and there is no comparable smaller-scale alternative available.

The results in Table 3 clearly show that using an extractor at least as strong as the reasoning model[6] leads to substantial improvements. For example, when the reasoning model is GPT-OSS-20B, pairing it with the stronger Qwen3-480B extractor yields the best performance, reaching 32.16% / 29.25%, compared to only 25.26% / 20.18% without extraction. Similarly, when the reasoning model is Qwen3-30B, the 480B extractor provides a gain of nearly 6 absolute points on both

[6]Since GPT-OSS-20B itself performs better than Qwen3-30B without extraction, we consider it the stronger of the two smaller models, consistent with the observed trends, despite having fewer parameters.

Table 3: Ablation study on the impact of different structure extractors. The multi-modal reasoning model is fixed with GLM-4.5V, while we vary the language reasoning model among Qwen3-480B, Qwen3-30B, and GPT-OSS-20B.

Model	Extractor	Soft	Hard
Qwen3-30B	Qwen3-480B	**22.37**	**18.42**
Qwen3-30B	Qwen3-30B	17.47	13.82
Qwen3-30B	GPT-OSS-20B	19.44	15.79
Qwen3-30B	-	16.41	12.83
GPT-OSS-20B	Qwen3-480B	**32.16**	**29.25**
GPT-OSS-20B	Qwen3-30B	25.71	20.83
GPT-OSS-20B	GPT-OSS-20B	27.85	22.59
GPT-OSS-20B	-	25.26	20.18

metrics. Interestingly, the effect is asymmetric: a weak reasoner benefits strongly from a powerful extractor, but a strong reasoner gains little from a weak extractor. For instance, GPT-OSS-20B with a Qwen3-30B extractor (25.71% / 20.83%) is close to its no-extraction baseline, suggesting that noisy structural cues are largely ignored or overridden by the reasoning model's own analysis. In contrast, Qwen3-30B paired with the 480B extractor enjoys a substantial jump, confirming that accurate structural sketches act as an external scaffold that smaller models can leverage.

4.3.3 Efficiency Analysis

We further analyze the computational efficiency of our framework in terms of runtime, and the number of agent calls. Across spreadsheets, our method requires 97.49 seconds on average for end-to-end extraction, with 19.54 agent calls per spreadsheet. The maximum observed memory usage is 21 GB. These results indicate that the computational cost of our framework is mainly reflected in iterative agent interactions rather than memory consumption. This suggests that the proposed extraction-verification pipeline is efficient enough for real-world spreadsheet understanding scenarios, while remaining independent of the specific query content.

5 Conclusion

In this paper, we presented SpreadsheetAgent, a two-stage multi-agent framework for spreadsheet understanding. By incrementally exploring localized regions and constructing structural sketches, SpreadsheetAgent preserves layout semantics while overcoming context-length limitations. With the integration of code execution, image conversion, and LaTeX conversion, the framework effectively captures numerical, visual, and structural cues. Experiments on SpreadsheetBench show that SpreadsheetAgent achieves significant gains over prior methods, and ablation studies verify the necessity of verification, structured sketching, and multi-format tools. Future work includes exploring adaptive strategies for tool invocation and region selection in real-world applications.

Limitations

Although our framework demonstrates promising results in spreadsheet understanding, several limitations remain to be addressed in future work. First, our approach inherently depends on the capabilities of existing foundation models, such as GLM-4.5V, which simultaneously support multimodal inputs (e.g., images) and function calls. These models provide the technical backbone that enables our multi-agent design. However, such strong and versatile models are still relatively scarce in the open-source community, which limits reproducibility and the potential for wide adoption. Exploring how to adapt the framework to more accessible or specialized open-source models would be an important step toward broader applicability. Second, this paper does not investigate model distillation, namely transferring the reasoning and conversational traces of large models into smaller ones. A key obstacle is the lack of large-scale, realistic spreadsheet datasets that can support effective distillation and evaluation. Without such resources, it remains challenging to systematically assess whether smaller models can inherit the reasoning capabilities of larger ones. We leave this as an important direction for future work.

Ethics Statement

The models utilized in this paper, Qwen3[7] , GPT-OSS[8] , and GLM-4.5V[9] , are open-sourced and licensed strictly for academic research purposes. Their use in this work adheres to the terms specified by the respective licenses, and no commercial deployment or redistribution of these models has been performed. Similarly, the datasets employed in our study, Spreadsheet Bench[10] and RealHiT

[7] https://github.com/QwenLM/Qwen3
[8] https://github.com/openai/gpt-oss
[9] https://github.com/zai-org/GLM-V
[10] https://github.com/RUCKBReasoning/SpreadsheetBench

Bench[11] , are also made available exclusively for research purposes, and all experiments were conducted in accordance with these intended uses.

Acknowledgment

This project is funded in part by Shenzhen Loop Area Institute, by the Centre for Perceptual and Interactive Intelligence (CPII) Ltd under the Innovation and Technology Commission (ITC)'s InnoHK, in part by NSFC-RGC Project N_CUHK498/24, and in part by Guangdong Basic and Applied Basic Research Foundation (No. 2023B1515130008, XW).

References

Lang Cao. 2025. Tablemaster: A recipe to advance table understanding with language models. *CoRR*, abs/2501.19378.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2020. Tabfact: A large-scale dataset for table-based fact verification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, Jianye Hao, Hangyu Mao, and Fuzheng Zhang. 2025. Sheetagent: Towards a generalist agent for spreadsheet reasoning and manipulation via large language models. In *Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025*, pages 158–177. ACM.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022. Hitab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1094–1110. Association for Computational Linguistics.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, and S. S. Li. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, Baoxu Wang, Boyan Shi, Changyu Pang, Chenhui Zhang, Da Yin, Fan Yang, Guoqing Chen, Jiazheng Xu, Jiali Chen, Jing Chen, Jinhao Chen, Jinghao Lin, Jinjiang Wang, Junjie Chen, Leqi Lei, Letian Gong, Leyi Pan, Mingzhi Zhang, Qinkai Zheng, Sheng Yang, Shi Zhong, Shiyu Huang, Shuyuan Zhao, Siyan Xue, Shangqin Tu, Shengbiao Meng, Tianshu Zhang, Tianwei Luo, Tianxiang Hao, Wenkai Li, Wei Jia, Xin Lyu, Xuancheng Huang, Yanling Wang, Yadong Xue, Yanfeng Wang, Yifan An, Yifan Du, Yiming Shi, Yiheng Huang, Yilin Niu, Yuan Wang, Yuanchang Yue, Yuchen Li, Yutao Zhang, Yuxuan Zhang, Zhanxiao Du, Zhenyu Hou, Zhao Xue, Zhengxiao Du, Zihan Wang, Peng Zhang, Debing Liu, Bin Xu, Juanzi Li, Minlie Huang, Yuxiao Dong, and Jie Tang. 2025. Glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *CoRR*, abs/2507.01006.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. *CoRR*, abs/2401.04088.

Rihui Jin, Zheyu Xin, Xing Xie, Zuoyi Li, Guilin Qi, Yongrui Chen, Xinbang Dai, Tongtong Wu, and Gholamreza Haffari. 2025. Table-r1: Self-supervised and reinforcement learning for program-based table reasoning in small language models. *CoRR*, abs/2506.06137.

Thomas Joshi, Herman Saini, Neil Dhillon, Antoni Viros i Martin, and Kaoutar El Maghraoui. 2025. Paged attention meets flexattention: Unlocking long-context efficiency in deployed inference. *CoRR*, abs/2506.07311.

[11] https://github.com/cspzyy/RealHiTBench

Yannis Katsis, Saneem A. Chemmengath, Vishwajeet Kumar, Samarth Bharadwaj, Mustafa Canim, Michael R. Glass, Alfio Gliozzo, Feifei Pan, Jaydeep Sen, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2022. AIT-QA: question answering dataset over complex tables in the airline industry. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track, NAACL 2022, Hybrid: Seattle, Washington, USA + Online, July 10-15, 2022*, pages 305–314. Association for Computational Linguistics.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. 2023. Sheetcopilot: Bringing software productivity to the next level through large language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-gpt: Table fine-tuned GPT for diverse table tasks. *Proc. ACM Manag. Data*, 2(3):176.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1470–1480. The Association for Computer Linguistics.

Houxing Ren, Zimu Lu, Weikang Shi, Haotian Hou, Yunqiao Yang, Ke Wang, Aojun Zhou, Junting Pan, Mingjie Zhan, and Hongsheng Li. 2025a. Alignment with fill-in-the-middle for enhancing code generation.

Houxing Ren, Mingjie Zhan, Zhongyuan Wu, and Hongsheng Li. 2024. Empowering character-level text infilling by eliminating sub-tokens.

Houxing Ren, Mingjie Zhan, Zhongyuan Wu, Aojun Zhou, Junting Pan, and Hongsheng Li. 2025b. Reflectioncoder: Learning from reflection sequence for enhanced one-off code generation.

Weikang Shi, Houxing Ren, Junting Pan, Aojun Zhou, Ke Wang, Zimu Lu, Yunqiao Yang, Yuxuan Hu, Linda Wei, Mingjie Zhan, and Hongsheng Li. 2026. From solver to tutor: Evaluating the pedagogical intelligence of llms with kmp-bench.

Weikang Shi, Aldrich Yu, Rongyao Fang, Houxing Ren, Ke Wang, Aojun Zhou, Changyao Tian, Xinyu Fu, Yuxuan Hu, Zimu Lu, Linjiang Huang, Si Liu, Rui Liu, and Hongsheng Li. 2025. Mathcanvas: Intrinsic visual chain-of-thought for multimodal mathematical reasoning.

Aofeng Su, Aowen Wang, Chao Ye, Chen Zhou, Ga Zhang, Gang Chen, Guangcheng Zhu, Haobo Wang, Haokai Xu, Hao Chen, Haoze Li, Haoxuan Lan, Jiaming Tian, Jing Yuan, Junbo Zhao, Junlin Zhou, Kaizhe Shou, Liangyu Zha, Lin Long, Liyao Li, Pengzuo Wu, Qi Zhang, Qingyi Huang, Saisai Yang, Tao Zhang, Wentao Ye, Wufang Zhu, Xiaomeng Hu, Xijun Gu, Xinjie Sun, Xiang Li, Yuhang Yang, and Zhiqing Xiao. 2024. Tablegpt2: A large multimodal model with tabular data integration. *CoRR*, abs/2411.02059.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets LLM: can large language models understand structured table data? A benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining, WSDM 2024, Merida, Mexico, March 4-8, 2024*, pages 645–654. ACM.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Pengzuo Wu, Yuhang Yang, Guangcheng Zhu, Chao Ye, Hong Gu, Xu Lu, Ruixuan Xiao, Bowen Bao,

Yijing He, Liangyu Zha, Wentao Ye, Junbo Zhao, and Haobo Wang. 2025a. Realhitbench: A comprehensive realistic hierarchical table benchmark for evaluating llm-based table analysis. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 7105–7137. Association for Computational Linguistics.

Zhenhe Wu, Jian Yang, Jiaheng Liu, Xianjie Wu, Changzai Pan, Jie Zhang, Yu Zhao, Shuangyong Song, Yongxiang Li, and Zhoujun Li. 2025b. Table-r1: Region-based reinforcement learning for table understanding. *CoRR*, abs/2505.12415.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jian Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 technical report. *CoRR*, abs/2505.09388.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Xuanliang Zhang, Dingzirui Wang, Keyan Xu, Qingfu Zhu, and Wanxiang Che. 2025. Rot: Enhancing table reasoning with iterative row-wise traversals. *CoRR*, abs/2505.15110.

Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M. Patel. 2024. Reactable: Enhancing react for table question answering. *Proc. VLDB Endow.*, 17(8):1981–1994.

Weichao Zhao, Hao Feng, Qi Liu, Jingqun Tang, Binghong Wu, Lei Liao, Shu Wei, Yongjie Ye, Hao Liu, Wengang Zhou, Houqiang Li, and Can Huang. 2024a. Tabpedia: Towards comprehensive visual table understanding with concept synergy. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Yilun Zhao, Lyuhao Chen, Arman Cohan, and Chen Zhao. 2024b. Tapera: Enhancing faithfulness and interpretability in long-form table QA by content planning and execution-based reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 12824–12840. Association for Computational Linguistics.

Mingyu Zheng, Xinwei Feng, Qingyi Si, Qiaoqiao She, Zheng Lin, Wenbin Jiang, and Weiping Wang. 2024. Multimodal table understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9102–9124. Association for Computational Linguistics.

Bangbang Zhou, Zuan Gao, Zixiao Wang, Boqiang Zhang, Yuxin Wang, Zhineng Chen, and Hongtao Xie. 2025. Syntab-llava: Enhancing multimodal table understanding with decoupled synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pages 24796–24806. Computer Vision Foundation / IEEE.

Ruiyan Zhu, Xi Cheng, Ke Liu, Brian Zhu, Daniel Jin, Neeraj Parihar, Zhoutian Xu, and Oliver Gao. 2025. Sheetmind: An end-to-end llm-powered multi-agent framework for spreadsheet automation. *CoRR*, abs/2506.12339.

Appendix

A Additional Experiments

A.1 RealHiT Bench

Baseline. In the RealHiT Bench experiments, we establish two primary categories of baselines. The first category consists of models that directly ingest the LaTeX tables as plain text inputs. Since the tables in this benchmark are relatively small, such models can process the entire structure in a single pass without exceeding context limits. However, this approach fails to capture the strengths of stepwise exploration and overlooks many structural and stylistic cues that become crucial in more complex, real-world spreadsheets. The second category of baselines involves models that operate over file inputs and attempt step-by-step reasoning, simulating an incremental reading process. To strengthen the comparison, we also include TreeThinker as a representative method for structured chain-of-thought reasoning.

LaTeX input. As shown in Table 6, when RealHiT tables are directly provided in LaTeX format, both TreeThinker and SpreadsheetAgent offer noticeable improvements on smaller models, while the benefits diminish for stronger backbones. For example, on Llama3.1-8B, SpreadsheetAgent increases Fact Checking EM from 33.20 to 34.51 and Numerical Reasoning EM from 10.38 to 15.69, outperforming TreeThinker in all three sub-tasks. A similar trend is observed on Qwen2.5-7B, where SpreadsheetAgent achieves 51.27 EM in Fact Checking and 32.30 EM in Numerical Reasoning, surpassing both the plain re-evaluated baseline and TreeThinker. However, for larger models such as Llama3.3-70B and Qwen2.5-72B, the improvements become marginal. For instance, SpreadsheetAgent only yields a +0.65 EM gain on Numerical Reasoning with Qwen2.5-72B, while showing slight regressions on Structure Comprehending. These results suggest that when the backbone itself is sufficiently strong, direct LaTeX input already enables effective reasoning, leaving limited headroom for additional exploration mechanisms.

File input. In contrast, as shown in Table 7, when feeding models with spreadsheet files and enabling incremental exploration, both TreeThinker and SpreadsheetAgent bring consistent gains across all backbones, and the improvements from SpreadsheetAgent are substantially larger. For example, on GPT-OSS-20B, SpreadsheetAgent raises Fact Checking EM by +5.92 and Numerical Reasoning EM by +9.99 compared to the plain baseline, outperforming TreeThinker by wide margins. On Qwen3-30B, SpreadsheetAgent delivers even more pronounced benefits, improving Fact Checking from 37.63 (TreeThinker) to 46.18 EM and Numerical Reasoning from 31.26 to 35.80 EM. Importantly, these advantages also extend to very large models: with Qwen3-Coder-480B, SpreadsheetAgent reaches 69.10 EM / 77.18 F1 in Fact Checking and 76.26 EM / 82.71 F1 in Structure Comprehending, while TreeThinker suffers severe performance drops. These findings confirm that step-by-step file exploration with multi-format inputs is crucial, as it provides systematic gains across scales and tasks, demonstrating that SpreadsheetAgent can robustly handle complex spreadsheet reasoning beyond the limits of text-only representations.

A.2 Case Study

Here, we present a representative case to illustrate the effectiveness of the proposed method, shown in Figure 2. The generated codes from both the baseline method and our proposed method are presented in Table 4. In this case, the baseline method failed because it mistakenly applied the employee filter to column L of the detail rows, rather than the correct column A (`User`). Since column L in the detail area is mostly empty, almost all rows were ignored, and the totals degenerated into trivial values close to zero. A snippet of the problematic code is shown below:

```
# BUG: employee filter is applied to col
    L (index 11)
df.iloc[i, 11] == employee

# Correct condition should be:
df.iloc[i, 0] == employee
```

This error highlights a fundamental limitation of text-only approaches: without explicit structural awareness, the model cannot reliably distinguish between header rows, detail areas, and summary regions. As a result, it may select the wrong reference column or misalign the header–data correspondence.

By contrast, with structure extraction, our method explicitly recognizes that the spreadsheet contains two semantically distinct regions:

```
### Main Table: Leave Entries
```yaml
...
```
```

Question

How can I calculate the total hours each employee has spent on each category by summing the values from column H when column L matches the corresponding employee's name in column A and the category matches with row 4?

Input Spreadsheet

Leave used summary from PP 2020-20 (Sep 27, 2020) to PP 2021-15 (July 31, 2021)

User	PP	Date	Type	From	To	Length	Length in Hours
Employee 1	21	Tue 13-Oct-20	Annual Leave	7:30am	4:00pm	8:00 Hours	8.00
Employee 1	21	Wed 14-Oct-20	Annual Leave	7:30am	4:00pm	8:00 Hours	8.00
Employee 1	21	Thu 15-Oct-20	Annual Leave	7:30am	4:00pm	8:00 Hours	8.00
Employee 1	21	Fri 16-Oct-20	Annual Leave	7:30am	4:00pm	8:00 Hours	8.00
Employee 1	23	Mon 16-Nov-20	Annual Leave	7:30am	4:00pm	8:00 Hours	8.00

what is the formula to sum total for each employee under each category?

	Annual Leave	LWOP	Sick Leave	Compensatory Time	Court Leave	AWOL	Other Leave	Military Leave	Military DC Nat Guard Leave
Employee 1									
Employee 2									
Employee 3									
Employee 4									

Figure 2: Case 1: Partial view of the leave entries spreadsheet.

Question

How can I write a formula in Excel for a forecast document where the revenue calculated by multiplying average order value by number of sales is adjusted by applying a percentage figure in row 13, but only if row 13 is not blank; otherwise, the original calculation should be used?

Input Spreadsheet

		January	February	March	April	May	June
Number of Sales		2000	1000	3000	6000	9000	5000
AOV		15.21	15.92	19.97	21.04	20.52	21.25
Revenue		30420	15920	59910	126240	184680	106250

				Manual Adjustments in %			
		January	February	March	April	May	June
Revenue			5.00%		10.00%		

Figure 3: Case 2: Partial view of the sales data spreadsheet.

```
### Designated Summary Area
```yaml
...
```

The exploration agent identifies the main data table (leave entries) as the core source of employee-level records, while also detecting the designated summary area as a separate, higher-level aggregation. This structural sketch prevents the misinterpretation of empty or auxiliary columns as valid keys. Moreover, because our framework incorporates layout cues and region-based parsing, the model correctly aligns headers with the corresponding data fields. It then applies the filter condition to column A (`User`), thereby including all valid rows and yielding accurate totals.

Another case is shown in Figure 3. The generated codes from both the baseline method and our proposed method are presented in Table 5. In this case, the baseline method failed because it mistakenly wrote the computed answer into line 9, which is outside the actual table boundary. A snippet of the problematic code is shown below:

```
df.iloc [8, col] = adjusted_revenue
```

This error occurred because the baseline approach lacked an explicit understanding of the spreadsheet's structural boundaries.

By contrast, our method leverages structure extraction to explicitly recognize the table boundaries. In this example, the exploration agent identified that the core data table is constrained to the range B5:H8 (noting that spreadsheet notation starts at 1, whereas arrays are 0-indexed). This structured annotation is recorded as:

```
node_3:
  start_index: "B8"
  end_index: "B8"
  value: "Revenue"
  children: []
```

With this structured context, the model becomes explicitly aware of the target location. It understands that the adjusted revenue should be placed in row 8, and any attempt to place it outside this range would lead to an inconsistency with the spreadsheet layout. As a result, the model ensures that the answer is correctly filled into the designated cell in row 8, avoiding any misplacement.
```

Table 4: Generated codes for Case 1 (leave entries).

```
# Code Generated by Qwen3-Coder-480B
import pandas as pd
from openpyxl import load_workbook

df = pd.read_excel('/mnt/data/input/1_51859_input.xlsx', header=None)

employees = ['Employee 1', 'Employee 2', 'Employee 3', 'Employee 4']

categories = []
for i in range(12, min(21, len(df.columns))):
    if pd.notna(df.iloc[3, i]):
        categories.append(df.iloc[3, i])
results = {}

for employee in employees:
    results[employee] = {}
    for category in categories:
        total_hours = 0
        for i in range(4, len(df)):
            if (pd.notna(df.iloc[i, 11]) and
                df.iloc[i, 11] == employee and
                pd.notna(df.iloc[i, 3]) and
                df.iloc[i, 3] == category and
                pd.notna(df.iloc[i, 7])):
                total_hours += df.iloc[i, 7]
        results[employee][category] = total_hours

wb = load_workbook('/mnt/data/input/1_51859_input.xlsx')
ws = wb.active

for i, employee in enumerate(employees):
    row_index = 5 + i
    for j, category in enumerate(categories):
        col_index = 13 + j
        ws.cell(row=row_index, column=col_index, value=results[employee][category])

wb.save('/mnt/data/output/1_51859_output.xlsx')
```

```
# Code Generated by Qwen3-Coder-480B w/ StructureAgent
import pandas as pd

file_path = "/mnt/data/input/1_51859_input.xlsx"
df = pd.read_excel(file_path, sheet_name="Leave used by employee", header=None)

data_rows = df.iloc[4:194, [0, 3, 7]]
data_rows.columns = ['User', 'Type', 'Hours']

data_rows['Hours'] = pd.to_numeric(data_rows['Hours'], errors='coerce').fillna(0)

expected_employees = df.iloc[4:8, 11].dropna().tolist()
expected_leave_types = df.iloc[3, 12:21].tolist()

summary_table = pd.pivot_table(data_rows, values='Hours', index='User', columns='
    Type', aggfunc='sum', fill_value=0)

summary_table = summary_table.reindex(index=expected_employees, columns=
    expected_leave_types, fill_value=0)

for i, employee in enumerate(expected_employees):
    for j, leave_type in enumerate(expected_leave_types):
        row_idx = 4 + i
        col_idx = 12 + j
        df.iloc[row_idx, col_idx] = summary_table.loc[employee, leave_type]

output_path = "/mnt/data/output/1_51859_output.xlsx"
df.to_excel(output_path, sheet_name="Leave used by employee", header=False)
```

Table 5: Generated codes for Case 2 (sales data).

```python
# Code Generated by Qwen3-Coder-480B
import pandas as pd

df = pd.read_excel('/mnt/data/input/1_52888_input.xlsx', header=None)

for col in range(2, 8):
    num_sales = df.iloc[5, col]
    aov = df.iloc[6, col]
    adjustment = df.iloc[12, col]

    base_revenue = num_sales * aov

    if pd.notna(adjustment):
        adjusted_revenue = base_revenue * (1 + adjustment)
    else:
        adjusted_revenue = base_revenue

    df.iloc[8, col] = adjusted_revenue

df.to_excel('/mnt/data/output/1_52888_output.xlsx', index=False, header=False)
```

```python
# Code Generated by Qwen3-Coder-480B w/ StructureAgent
import pandas as pd

df = pd.read_excel('/mnt/data/input/1_52888_input.xlsx', header=None)

number_of_sales = df.iloc[5, 2:8].values
aov = df.iloc[6, 2:8].values

base_revenue = number_of_sales * aov

adjustments = df.iloc[12, 2:8].values

adjusted_revenue = []
for i in range(len(base_revenue)):
    if pd.notna(adjustments[i]) and adjustments[i] != 0:
        adjusted_value = base_revenue[i] * (1 + adjustments[i])
        adjusted_revenue.append(adjusted_value)
    else:
        adjusted_revenue.append(base_revenue[i])

df.iloc[7, 2:8] = adjusted_revenue

df.to_excel('/mnt/data/output/1_52888_output.xlsx', sheet_name='Sheet1', index=False
    , header=False)
```

Table 6: Performance comparison across different models on three sub-tasks of RealHiT Bench: Fact Checking, Numerical Reasoning, and Structure Comprehending. Results marked with * indicate re-evaluations under a slightly modified prompt that encourages models to perform explicit reasoning before producing final answers, as opposed to directly outputting results.

Model	Fact Checking		Numerical Reasoning		Structure Comprehending	
	EM	F1	EM	F1	EM	F1
GPT4o	60.31	68.97	38.65	50.12	63.04	71.14
Gemini1.5-pro	59.08	66.14	35.54	43.74	63.64	69.71
DeepSeek-R1	70.91	79.45	70.31	72.54	82.71	84.62
Llama3.1-8B-Instruct	30.32	44.93	14.53	27.21	35.90	50.80
Llama3.1-8B-Instruct*	33.20	44.15	10.38	22.58	25.51	39.51
w/ TreeThinker	32.37	44.93	13.36	24.76	21.72	38.78
w/ SpreadsheetAgent	**34.51**	**47.68**	**15.69**	**28.36**	**25.25**	**40.33**
Qwen2.5-7B-Instruct	18.65	38.39	5.32	19.75	23.48	44.81
Qwen2.5-7B-Instruct*	48.56	54.70	29.83	38.29	45.96	55.45
w/ TreeThinker	51.19	57.45	31.13	39.14	44.95	53.63
w/ SpreadsheetAgent	**51.27**	**57.56**	**32.30**	**40.44**	**48.74**	**56.61**
Llama3.3-70B-Instruct	53.08	64.53	36.58	48.99	55.81	68.93
Llama3.3-70B-Instruct*	67.13	**73.14**	54.47	61.60	73.99	78.61
w/ TreeThinker	**67.30**	73.06	57.98	**64.92**	**74.24**	**79.49**
w/ SpreadsheetAgent	65.74	71.94	**58.37**	64.44	**74.24**	79.36
Qwen2.5-72B-Instruct	51.93	62.15	26.98	39.23	54.55	68.34
Qwen2.5-72B-Instruct*	65.74	72.93	52.40	61.33	**73.48**	**80.54**
w/ TreeThinker	**66.80**	**74.03**	50.97	60.33	72.73	79.45
w/ SpreadsheetAgent	66.64	73.44	**53.05**	**62.31**	70.71	78.42

Table 7: Performance comparison across different models with Python tool on three sub-tasks of RealHiT Bench: Fact Checking, Numerical Reasoning, and Structure Comprehending.

Model	Fact Checking		Numerical Reasoning		Structure Comprehending	
	EM	F1	EM	F1	EM	F1
GPT-OSS-20B	42.81	48.54	34.63	39.03	33.84	40.46
w/ TreeThinker	45.11	50.12	39.82	44.42	33.59	39.53
w/ SpreadsheetAgent	**48.73**	**53.64**	**44.62**	**48.92**	**39.65**	**44.74**
GPT-OSS-120B	58.18	62.75	55.38	59.16	54.04	57.19
w/ TreeThinker	56.53	61.14	54.60	59.06	53.03	57.98
w/ SpreadsheetAgent	**64.09**	**69.22**	**58.24**	**62.58**	**62.37**	**66.01**
Qwen3-30B	36.24	42.31	28.66	35.76	37.88	44.26
w/ TreeThinker	37.63	45.03	31.26	39.77	34.09	41.65
w/ SpreadsheetAgent	**46.18**	**52.44**	**35.80**	**44.36**	**40.15**	**47.28**
Qwen3-235B	56.94	62.60	42.54	48.94	61.87	67.85
w/ TreeThinker	57.27	62.54	44.10	50.06	59.09	63.28
w/ SpreadsheetAgent	**62.70**	**68.55**	**47.47**	**54.25**	**63.89**	**69.43**
Qwen3-Coder-480B	66.47	74.46	54.60	64.97	72.98	80.01
w/ TreeThinker	57.85	65.87	47.21	58.05	58.33	65.77
w/ SpreadsheetAgent	**69.10**	**77.18**	**55.25**	**65.74**	**76.26**	**82.71**

Prompt for Structure Extraction

You are a spreadsheet expert who can analyze and manipulate spreadsheets using Python. Your task is to detect tables in a spreadsheet and generate hierarchical structures for their top-row and left-column headers.

(1) Detect All Tables
Identify all rectangular areas in the spreadsheet that can be considered tables. A table is defined as a block with identifiable headers and corresponding data. Clarify how to handle:
- Empty rows/columns (skip or stop).
- Multiple adjacent tables (detect separately).
- Merged cells (treat as higher-level headers).

(2) Analyze and Output Table Structure

a. Metadata
- Sheet Name: The sheet containing the table.
- Table Name: If defined (e.g., Table1 in Excel).
- Table Range: The full rectangular range of the table (e.g., A1:D20).
- Data Range: The rectangular range containing only data (excluding headers).
- Notes / Footnotes: Optional annotations or data source lines below the table.

b. Header Structure
- Determine Header Format
First, classify the table structure into one of the following:
- Column-only header (most common: header row(s) at the top)
- Row-only header (rare: header column(s) on the left)
- Both row & column headers (matrix-style tables)

- Detect all header cell:
Each header cell have three attributes: start position, end position, and value.
- Start Position: The coordinate of the top-left cell of the header (e.g., A1).
- End Position: The coordinate of the bottom-right cell of the header (e.g., B2).
- Value: The text content of the header cell.

- Group header cells with the same level:
Headers can be nested, indicating hierarchical relationships. Group headers based on their indentation levels or merged cell spans. For example, if "Country" spans two columns and "State" and "City" are under it, they should be grouped accordingly.

- Construct hierarchical structures:
Build the hierarchical structure for both row and column headers. Each node in the hierarchy should include:
- Start Index: The starting cell coordinate of the header.
- End Index: The ending cell coordinate of the header.
- Value: The text content of the header.
- Children: A list of child nodes representing sub-headers.

c. Data Properties
For each column (or row header if applicable), detect:
- Data Type (string, number, date, boolean, mixed)

- Unit (if explicit, e.g., km, dollar)
- Format (e.g., bold, italic, background color, currency)

(3) Output Format
For each table, please generate a hierarchical structure in the following YAML format (with your reasoning content and other summary infomation).

```yaml
sheet_name: sheet_name
table_name: table_name
table_range: table_range
data_range: data_range
notes:
  - ...
  - ...
row_header:
    node_1:
        start_index: A3
        end_index: A4
        value: City
        children:
            - node_1.1:
                start_index: B3
                end_index: B3
                value: Population
                children: ...
            - node_1.2:
                start_index: B4
                end_index: B4
                value: Area
                children: ...
column_header:
    node_1:
        start_index: C1
        end_index: D1
        value: Country
        children:
            - node_1.1:
                start_index: C2
                end_index: C2
                value: State
                children: ...
            - node_1.2:
                start_index: D2
                end_index: D2
                value: City
                children: ...
data_properties:
  Population:
    type: number
    unit: people
```

```
    format: plain
  Area:
    type: number
    unit: km
    format: comma-separated
```

spreadsheet_path
{spreadsheet_path}

sheet_name
{sheet_name}

used_range
{used_range}

Prompt for Verification

You are a Spreadsheet Structure Verification Assistant. Your task is to verify whether a spreadsheet's structure matches the expected description provided.

Instructions:
1. Utilize the tool to get the information and determine if the structure is **correct and consistent**.
2. If there are discrepancies (e.g., mismatched ranges, missing/extra rows or columns, incorrect headers, or formatting inconsistencies), describe them clearly and precisely.
3. If the structure is valid, confirm that no issues were found.

Output Format:
```yaml
verification: true/false,
issues:
    - "Description of issue 1",
    - "Description of issue 2"
```
- verification is true: Structure is correct, "issues" should be an empty list.
- verification is false: One or more issues found, list them in "issues".

Information to Verify:

- Spreadsheet Path: {spreadsheet_path}
- Sheet Name: {sheet_name}
- Used Range: {used_range}
- Spreadsheet Structure:
{spreadsheet_info}

Definition of Code Execution Tool

```
{
   "type": "function",
   "function": {
      "name": "execute_python",
      "description": "When you send a message containing Python code to python, it will be executed in a stateful Jupyter notebook environment. python will respond with the output of the execution or time out after 60.0 seconds. The drive at '/mnt/data' can be used to save and persist user files. Internet access for this session is disabled.",
      "parameters": {
         "type": "object",
         "properties": {
            "code": {
               "type": "string",
               "description": "The Python code to execute",
            }
         },
         "required": ["code"]
      },
   }
}
```

Definition of Vision Range Description Tool


```
{
  "type": "function",
  "function": {
    "name": "vision_question_answer",
    "description": "You can analyze a selected range of cells in a spreadsheet using vision capabilities. When asking, focus on small, well-defined questions and describe all relevant details. Don't just hand over your entire task to the tool.",
    "parameters": {
      "type": "object",
      "properties": {
        "path": {
          "type": "string",
          "description": "The file path to the spreadsheet containing the image."
        },
        "sheet_name": {
          "type": "string",
          "description": "The name of the sheet from which to extract the image."
        },
        "range": {
          "type": "string",
          "description": "Optional. The cell range to search for the image (e.g., 'A1:D20'). Use to limit extraction to a specific area. Images smaller than 8192×16384 pixels are supported, so keep the range within reasonable bounds."
        },
        "question": {
          "type": "string",
          "description": "The question to answer about the specified range of cells."
        }
      },
      "required": ["path", "sheet_name", "range", "question"]
    }
  }
}
```

Definition of Latex Range Description Tool


```
{
  "type": "function",
  "function": {
    "name": "latex_question_answer",
    "description": "You can analyze a selected range of cells in a spreadsheet using latex capabilities. When asking, focus on small, well-defined questions and describe all relevant details — don't just hand over your entire task to the tool.",
    "parameters": {
      "type": "object",
      "properties": {
        "path": {
          "type": "string",
          "description": "The file path to the Excel file."
        },
        "sheet_name": {
          "type": "string",
          "description": "The name of the sheet from which to extract the latex."
        },
        "range": {
          "type": "string",
          "description": "Optional. The cell range to search for the latex (e.g., 'A1:D20'). Use to limit extraction to a specific area."
        },
        "question": {
          "type": "string",
          "description": "The question to answer about the specified range of cells."
        }
      },
      "required": ["path", "sheet_name", "range", "question"]
    }
  }
}
```

Definition of Excel to Vision Tool

```
{
  "type": "function",
  "function": {
    "name": "convert_excel_to_image",
    "description": "Extracts an image from a specified spreadsheet file.",
    "parameters": {
      "type": "object",
      "properties": {
        "path": {
          "type": "string",
          "description": "The file path to the Excel file."
        },
        "sheet_name": {
          "type": "string",
          "description": "The name of the sheet from which to extract the image."
        },
        "range": {
          "type": "string",
          "description": "The cell range to search for the image (e.g., 'A1:D20'). Use to limit extraction to a specific area. Images smaller than 8192×16384 pixels are supported, so keep the range within reasonable bounds."
        }
      },
      "required": ["path", "sheet_name", "range"]
    }
  }
}
```

Definition of Excel to Latex Tool

```
{
   "type": "function",
   "function": {
      "name": "convert_excel_to_latex",
      "description": "Converts an Excel file to a LaTeX file.",
      "parameters": {
         "type": "object",
         "properties": {
            "path": {
               "type": "string",
               "description": "The file path to the Excel file."
            },
            "sheet_name": {
               "type": "string",
               "description": "The name of the sheet from which to extract the image."
            },
            "range": {
               "type": "string",
               "description": "The cell range to convert to LaTeX. Use to limit conversion to a specific area."
            }
         },
         "required": ["path", "sheet_name", "range"]
      }
   }
}
```

YAML Structure Example

Main Table: Leave Entries

```yaml
sheet_name: Leave used by employee
table_name: Leave Entries
table_range: A4:U194
data_range: A5:U194
notes:
  - "Title: Leave used summary from PP 2020-20 (Sep 27, 2020) to PP 2021-15 (July 31, 2021)"
  - "Note: what is the formula to sum total for each employee under each category?"
header_format: column-only
row_header:
  node_1:
    children: []
    end_index: A50
    start_index: A5
    value: Employee 1
  node_2:
    children: []
    end_index: A98
    start_index: A51
    value: Employee 2
  node_3:
    children: []
    end_index: A158
    start_index: A99
    value: Employee 3
  node_4:
    children: []
    end_index: A194
    start_index: A159
    value: Employee 4
column_header:
  node_1:
    children: []
    end_index: A4
    start_index: A4
    value: User
  node_2:
    children: []
    end_index: B4
    start_index: B4
    value: PP
  node_3:
    children: []
    end_index: C4
    start_index: C4
    value: Date
  ...
```

````
data_properties:
  User:
    format: plain
    type: string
    unit: ”
  PP:
    format: plain
    type: string
    unit: ”
  Date:
    format: date-string
    type: date
    unit: ”
  ...
```

### Designated Summary Area

```yaml
sheet_name: Leave used by employee
table_name: Leave Summary Area
table_range: L4:U8
data_range: M5:U8
notes:
  - "This area is part of the main table but designated for summary calculations"
  - "Intended for formula-based calculations by employee and leave type"
  - "Currently contains only row headers (L5:L8) and column headers (M4:U4)"
header_format: both-row-and-column
row_header:
  node_1:
    children: []
    end_index: L5
    start_index: L5
    value: Employee 1
  node_2:
    children: []
    end_index: L6
    start_index: L6
    value: Employee 2
  node_3:
    children: []
    end_index: L7
    start_index: L7
    value: Employee 3
  node_4:
    children: []
    end_index: L8
    start_index: L8
    value: Employee 4
column_header:
````

```
  node_1:
    children: []
    end_index: M4
    start_index: M4
    value: Annual Leave
  node_2:
    children: []
    end_index: N4
    start_index: N4
    value: LWOP
  node_3:
    children: []
    end_index: O4
    start_index: O4
    value: Sick Leave
  ...
data_properties:
  Annual Leave:
    format: decimal
    type: empty
    unit: hours
  LWOP:
    format: decimal
    type: empty
    unit: hours
  Sick Leave:
    format: decimal
    type: empty
    unit: hours
  ...
```

Verification Example

verification: false
issues:
- "The 'Leave Entries' table range is defined as A4:U194, but the actual data ends before row 194, leaving many blank rows at the end."
- "In the 'Leave Summary Area', the range is defined as L4:U8, but column L contains employee names instead of being part of the summary calculation area. This misaligns the row headers for the summary section."
- "Column headers for the 'Leave Entries' table are correctly present in row 4, but the 'Leave Summary Area' overlaps with the main table without clear structural separation."
- "Some cells in the 'Leave Entries' table contain unexpected values (e.g., employee names in column M), suggesting misplaced data or incorrect usage of summary columns (M-U)."
- "The note 'what is the formula to sum total for each employee under each category?' appears in row 3, indicating incomplete setup for summary calculations in the designated area."