Encoding Spreadsheets for Large Language Models

Haoyu Dong[*†], **Jianbo Zhao**[†‡], **Yuzhang Tian**[†‡], **Junyu Xiong**[†‡],
Mengyu Zhou, **Yun Lin**[‡], **José Cambronero**, **Yeye He**, **Shi Han**, **Dongmei Zhang**
Microsoft Corporation

Abstract

Spreadsheets are characterized by their extensive two-dimensional grids, flexible layouts, and varied formatting options, which pose significant challenges for large language models (LLMs). In response, we introduce SHEETENCODER, pioneering an efficient encoding method designed to unleash and optimize LLMs' powerful understanding and reasoning capability on spreadsheets. Initially, we propose a vanilla serialization approach that incorporates cell addresses, values, and formats. However, this approach was limited by LLMs' token constraints, making it impractical for most applications. To tackle this challenge, three innovative modules are proposed to compress spreadsheets effectively: structural-anchor-based compression, inverse index translation, and data-format-aware aggregation. It significantly improves performance in spreadsheet table detection task, outperforming the vanilla approach by 25.6% in GPT4's in-context learning setting. Moreover, fine-tuned LLM with SHEETENCODER has an average compression ratio of 25×, but achieves a state-of-the-art 78.9% F1 score, surpassing the best existing models by 12.3%, demonstrating that SHEETENCODER greatly boosts LLMs's performance on spreadsheet data.

1 Introduction

Spreadsheets are ubiquitous for data management and extensively utilized within platforms like Microsoft Excel and Google Sheets. Understanding spreadsheet layout and structure (Dong et al., 2019b; Gol et al., 2019; Hulsebos et al., 2019; Dou et al., 2018; Wang et al., 2021; Deng et al., 2022; Chen and Cafarella, 2014), a longstanding challenge for traditional models and is crucial for effective data analysis, data manipulation, and various real-world applications involving intelligent

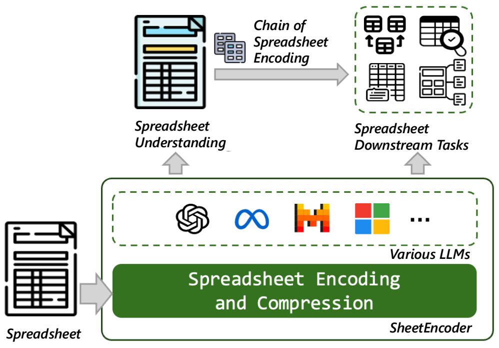

Chain of Spreadsheet Encoding
Spreadsheet Understanding
Spreadsheet Downstream Tasks
Various LLMs
Spreadsheet Encoding and Compression
SheetEncoder
Spreadsheet

Figure 1: LLMs for Spreadsheet with SHEETENCODER.

user interaction with spreadsheet data. Recently, the rapid development of Large Language Models (LLMs) has opened new frontiers in table processing (Li et al., 2023b) and reasoning (Cheng et al.). However, spreadsheets pose unique challenges for LLMs due to their expansive grids that usually exceed the token limitations of popular LLMs, as well as their inherent two-dimensional layouts and structures, which are poorly suited to linear and sequential input. Furthermore, LLMs often struggle with spreadsheet-specific features such as cell addresses and formats, complicating their ability to effectively parse and utilize spreadsheet data, as detailed in Appendix A.

In this paper, we introduce SHEETENCODER for LLMs, a pioneering encoding framework to unleash and maximize the potential of LLMs for spreadsheet understanding and reasoning. We initially propose a vanilla encoding method to serialize spreadsheets into sequences, augmenting the Markdown encoding method by including essential cell addresses and (optional) formats. Furthermore, large spreadsheets that exceed the token limits of LLMs not only limit their processing but also, as observed in prior studies, degrade accuracy performance as the size increases (Liu et al., 2024). To address this challenge, we equip SHEETENCODER with three portable modules to compress spreadsheet, including:

[*] Corresponding author.
[†] Equal contribution.
[‡] Work during internship at Microsoft.

1) Structural Anchors for Efficient Layout Understanding: Observations indicate that large spreadsheets often contain numerous homogeneous rows or columns, which contribute minimally to understanding the layout and structure (see left panel in Figure 2 (a)). To address this, we identify structural anchors—heterogeneous rows and columns at possible table boundaries that offer substantial layout insights, as depicted in Figure 2 (b). Then we remove distant, homogeneous rows and columns, producing a condensed "skeleton" version of the spreadsheet, as illustrated in Figure 2 (c).

2) Inverted-Index Translation for Token Efficiency: The vanilla encoding method becomes token-consuming when handling spreadsheets with numerous empty cells and repetitive values, as shown in Figure 2 (c). To improve efficiency, we depart from traditional row-by-row and column-by-column serialization and employ a lossless inverted-index translation in JSON format. This method creates a dictionary that indexes non-empty cell texts and merges addresses with identical text, optimizing token usage while preserving data integrity.

3) Data Format Aggregation for Numerical Cells: Adjacent numerical cells often share similar number formats. Recognizing that exact numerical values are less crucial for grasping spreadsheet structure, we extract number format strings and data types from these cells. Then adjacent cells with the same formats or types are clustered together. This method is visualized in the right example of Figure 2, where rectangular regions are represented by uniform format strings and data types, streamlining the understanding of numerical data distribution without excessive token expenditure.

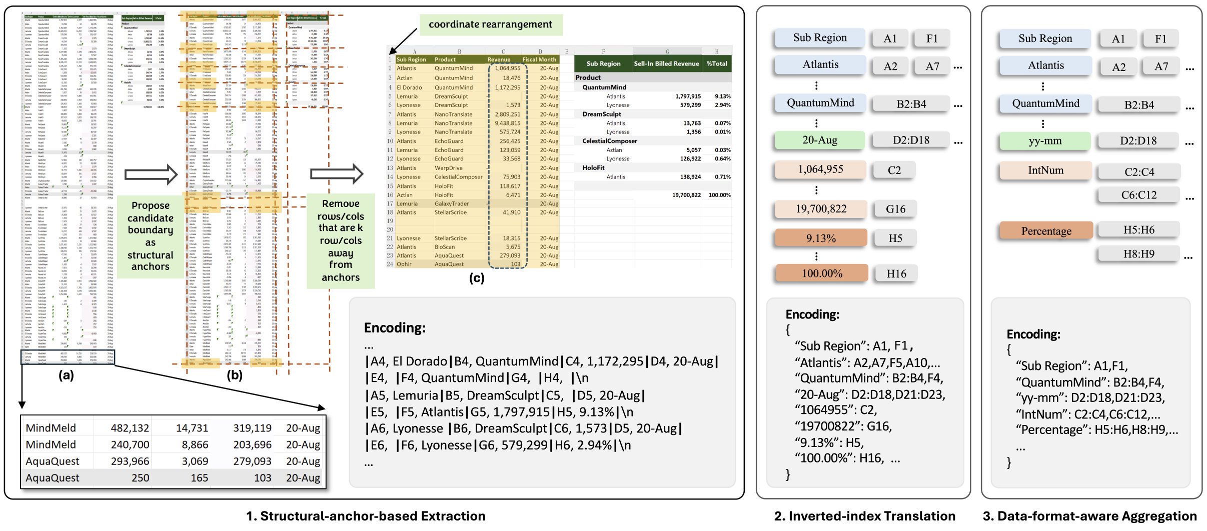
We conducted a comprehensive evaluation of our method on a variety of LLMs. Our experiments show that SHEETENCODER significantly reduces token usage for spreadsheet encoding by 96%. Moreover, SHEETENCODER has shown exceptional performance in spreadsheet table detection, the foundational task of spreadsheet understanding, surpassing TableSense-CNN by 12.3% (Dong et al., 2019b). We also applied SHEETENCODER to a representative spreadsheet QA task. Inspired by the Chain of Thought (CoT) methodology (Zheng et al., 2023; Jiang et al., 2023b), we propose Chain of Spreadsheet Encoding (CoS) to decompose spreadsheet reasoning into a table detection-match-reasoning pipeline. It significantly outperformed existing SOTA methods for table QA (Liu et al.; Cheng et al.). Our primary contributions are summarized as follows:

- We propose SHEETENCODER, the first work that substantially leverage LLMs for understanding and analyzing spreadsheet data. To address challenges in scale, diversity, and complexity of spreadsheets, three innovative modules are proposed to compress spreadsheets effectively: structural-anchor-based compression, inverse index translation, and data-format-aware aggregation.
- We fine-tune a variety of cutting-edge LLMs to achieve optimal performance on spreadsheet table detection, and demonstrate the high effectiveness of SHEETENCODER in accurately understanding complex spreadsheet layouts and structures.
- In order to extend the horizontal capabilities of SHEETENCODER to a wide range of downstream tasks, we propose CoS and verify it on Spreadsheet QA, highlighting its potential for intelligent user interaction.

2 Related Work

Spreadsheet Representation Spreadsheet representation involves converting the spreadsheets into specific representations for different models. There are various methods for spreadsheet (table) representation. (Dong et al., 2019a,b) enhance Mask-RCNN to leverage spatial and visual information in spreadsheets, and (Deng et al., 2024) explores the usage of LLMs to evaluate image tables, but it doesn't work well for spreadsheet images as input to VLMs (Xia et al., 2024). To capture sequential semantics in rows and columns, LSTMs are further adopted (Nishida et al., 2017; Gol et al., 2019) in row&column directions. Pre-trained LMs (Dong et al., 2022) are then proposed to understand spreadsheets (Wang et al., 2021). Recent studies (Zhang et al., 2024a; Li et al., 2023b; Sui et al., 2023) have explored the efficacy of using Markdown, HTML, JSON, and DFLoader for table representation, and these methods are comprehensively summarized in tutorial of (Dong and Wang, 2024). However, traditional table representations are not well suited to spreadsheets due to their assumption of single table input and absence of explicit cell addresses, as experiments show in Appendix B.

Spreadsheet Understanding While most table LLMs are restricted to single table settings, spreadsheets with multiple tables typically exceed token


coordinate rearrangement
Propose candidate boundary as structural anchors
Remove rows/cols that are k row/cols away from anchors
(a)
(b)
(c)
1. Structural-anchor-based Extraction
2. Inverted-index Translation
3. Data-format-aware Aggregation
Sub Region
Atlantis
QuantumMind
20-Aug
1,064,955
19,700,822
9.13%
100.00%
yy-mm
IntNum
Percentage

MindMeld	482,132	14,731	319,119	20-Aug
MindMeld	240,700	8,866	203,696	20-Aug
AquaQuest	293,966	3,069	279,093	20-Aug
AquaQuest	250	165	103	20-Aug

Encoding:
```
...
|A4, El Dorado|B4, QuantumMind|C4, 1,172,295|D4, 20-Aug|
|E4,  |F4, QuantumMind|G4,  |H4,  |\n
|A5, Lemuria|B5, DreamSculpt|C5,  |D5, 20-Aug|
|E5,  |F5, Atlantis|G5, 1,797,915|H5, 9.13%|\n
|A6, Lyonesse |B6, DreamSculpt|C6, 1,573|D5, 20-Aug|
|E6,  |F6, Lyonesse|G6, 579,299|H6, 2.94%|\n
...
```

Encoding:
```
{
  "Sub Region": A1, F1,
  "Atlantis": A2,A7,F5,A10,...
  "QuantumMind": B2:B4,F4,
  "20-Aug": D2:D18,D21:D23,
  "1064955": C2,
  "19700822": G16,
  "9.13%": H5,
  "100.00%": H16,  ...
}
```

Encoding:
```
{
  "Sub Region": A1,F1,
  "QuantumMind": B2:B4,F4,
  "yy-mm": D2:D18,D21:D23,
  "IntNum": C2:C4,C6:C12,...
  "Percentage": H5:H6,H8:H9,...
  ...
}
```

Figure 2: Illustration of the SHEETENCODER framework. The original spreadsheet contains two tables, featuring numerous data entries or hierarchical headers. The completed spreadsheet consists of 576 rows and 23 columns, with an vanilla encoding of 61,240 tokens. Initially, we first extract cells using structural anchors, rearranging them into a smaller 24×8 sheet. Subsequently, we perform index-invert, removing empty cells. Finally, we aggregate cells based one data formats, achieving an extremely compact representation of the spreadsheet, which contains only 708 tokens.

limits. Moreover, the diversity in multi-table layout and structure significantly confounds the problem. Spreadsheet table detection (Dong et al., 2019b; Christodoulakis et al., 2020; Doush and Pontelli, 2010; Vitagliano et al., 2022) aims at identifying all tables on a given sheet and determining their respective ranges. As a fundamental task for spreadsheet understanding, this task triggers hundreds of millions of daily average usage in commercial spreadsheet tools (Zhang et al., 2024b), and the accuracy still urges improvements due to the flexibility and complexity of spreadsheets.

Spreadsheet Downstream Tasks Spreadsheet understanding is enabling for a series of spreadsheet tasks for real world applications, such as table question answering analysis (He et al., 2024; Cheng et al., 2022b; Cheng et al.; Jiang et al., 2022; Liu et al.), table extraction (Chen and Cafarella, 2013, 2014; Li et al., 2024), formula or code generation (Chen et al., 2021; Cheng et al., 2022a; Joshi et al., 2024; Chen et al., 2024; Li et al., 2023a), error detection (Wang and He, 2019; Dou et al., 2016), etc. In this paper, we choose spreadsheet QA, one of the most demanded spreadsheet analysis tasks. It is an extension of the Table QA task in spreadsheet data, with the additional complexity of detecting and matching multiple tables within a spreadsheet.

LLMs' Token Efficiency Related work suggests that the performance of LLMs degrades significantly with long contexts (Liu et al., 2024; Xu et al., 2023). Efforts to improve model performance and reduce costs have led to the development of compression techniques for long prompts. Some researchers employ information-theory metrics to filter out redundant information (Li, 2023; Jiang et al., 2023a). Additionally, specialized models have been proposed to optimize prompt compression (Pan et al., 2024). However, these strategies primarily address natural language prompts and may not suit tabular data, potentially leading to considerable structure and data information loss. DBCopilot (Wang et al., 2023) enables text-to-SQL conversion on large databases through schema routing. However, due to LLMs' insufficient ability in understanding inherent multi-table layouts and complex table structures that cannot execute queries similar to SQL, schema routing is impractical, restricting the broader application of cutting-edge tabular works (Cheng et al.; Li et al., 2023b; Sui et al., 2024) on spreadsheet data.

3 Method

We propose a novel spreadsheet encoding framework in a Markdown-like style as text. To achieve a more compact and efficient representation, we introduce three independent yet combinable modules: structural-anchor-based extraction, inverted-index translation, and data-format-aware aggregation, which enable efficient data compression and enhance performance on downstream tasks.

3.1 Vanilla Spreadsheet Encoding with Cell Value, Address, and Format

Due to the absence of standardized practices in spreadsheet encoding for LLMs, we first explore traditional spreadsheet encoding methods. Appendix B presents a comparison of different mainstream tabular data encoding methods, including HTML, XML, and Markdown. Based on the encoding length and performance on spreadsheet understanding tasks, we use a Markdown-like style representation:

$$\mathcal{S} = \{Cell_{i,j}\}_{i \in m, j \in n}, \tag{1}$$

$$\begin{aligned}\mathcal{T} &= \text{markdown}\,\{\text{encode}\,(Cell_{i,j})\} \\ &:= \text{“}|Address_{i,j}, Value_{i,j}, Format|...\ \backslash n\text{”},\end{aligned} \tag{2}$$

where $\mathcal{S} \in \mathbb{R}^{m,n}$ denotes the spreadsheet, $\mathcal{T} \in \mathbb{R}^1$ denotes the text representation of a cell, and i, j, m, n respectively represent the row and column index of the cell and the row and column range of $\mathcal{S}$. We also explored the inclusion of cell format information (such as background color, bold font, borders, etc.) into each cell's representation. However, these experiments demonstrated that such detailed encoding adversely affects model performance due to rapid token limit exceedance and LLMs' inadequate capability to process format information effectively, as detailed in Appendix A. We plan to further explore this in future research, focusing on enhancing the model's ability to understand and utilize format and structural cues.

3.2 Structural-anchor-based Extraction

Large spreadsheets often feature numerous homogeneous rows or columns, which minimally contribute to the understanding of their layout and structure, as depicted in Figure 2 (a). To effectively compress spreadsheets while preserving vital layout and structural information, we propose a novel heuristic-based method, detailed further in Appendix C. This method identifies heterogeneous rows and columns at the edges of table boundaries—termed structural anchors:

$$\mathcal{A} = \{r_p, c_q\}_{p \in m, q \in n}, \tag{3}$$

where $r_p = \{Cell_{i,j}\}_{i=p, j \in n}$ and $c_q = \{Cell_{i,j}\}_{i \in m, j=q}$. Using these anchor points, our method discards rows and columns that are located more than k units away from any anchor point, because they rarely serve as table boundaries. The parameter k serves as a threshold to control the scope of neighborhood retention, effectively eliminating areas predominantly filled with homogeneous data that do not contribute to an understanding of the spreadsheet's layout and structure. We explored the effects of different k values in an ablation study, as detailed in Appendix D.1.

The extracted rows and columns can be expressed as:

$$\mathcal{A}_+ = \{r_{p+}, c_{q+}\}_{p+ \in m, q+ \in n}, \tag{4}$$

where the extracted "skeletons" are defined as: $r_{p+} = \{Cell_{i,j}\}_{|i-p| \le k, j \in n}$ and $c_{q+} = \{Cell_{i,j}\}_{i \in m, |j-q| \le k}$. Then we obtain the extracted compact spreadsheet:

$$\mathcal{S}_e = \text{extract}(\mathcal{S}) = \text{address\_map}(r_{p+} \cap c_{q+}). \tag{5}$$

Based on the compressed spreadsheet $\mathcal{S}_e$, we can obtain extremely shorter text representation $\mathcal{T}_e$. Furthermore, after extraction, we perform a coordinate re-mapping to ensure continuity in cell coordinates, preserving the integrity of data relationships within the compressed spreadsheet. This re-mapping is critical for maintaining the accuracy of prediction results, ensuring that analyses remain consistent even after compression. This method filters out 75% spreadsheet content but preserves 97% rows and columns at the edges of table boundaries.

3.3 Inverted-index Translation

Spreadsheets often contain numerous empty rows, columns, and scattered cells. The standard encoding method, as detailed in Section 3.1, employs a grid-based method that pairs cell addresses with their contents. This approach necessitates recording empty cells to maintain the spreadsheet's two-dimensional structure, which significantly increases token consumption. Furthermore, cells with identical values are encoded repeatedly, further exacerbating token usage.

To address these inefficiencies, we propose a two-stage Inverted-index-based Translation method. The first stage involves converting the traditional matrix-style encoding into a dictionary format, where cell values serve as keys indexing the addresses. In the second stage, cells sharing the same value are merged, with empty cells excluded and cell addresses noted as ranges. This method effectively reduces the number of required tokens by eliminating redundancies and simplifying the representation of repeated and empty cells. The translation process is represented mathematically as follows:

$$\begin{aligned}\mathcal{T}_t &= \text{invert}(\mathcal{T})\\ &:= \{\mathit{Value} : \mathit{Address\ or\ Address\_Region}, ...\}. \quad (6)\end{aligned}$$

Inverted-index Translation is a **lossless compression** method general for all spreadsheet understanding tasks, and it remarkably increases SHEETENCODER's compression ratio from 4.41 to 14.91. More details can be found in Table 1.

3.4 Data-format-aware Aggregation

In spreadsheets, adjacent cells typically share the same data format. As shown in Figure 2 (3), column C records the sell-in billed revenue for different products. Nonetheless, the concrete numerical values are not essential for understanding the structure and semantics of the spreadsheet (although there might loss of fine-trained details of exact quantities, e.g., "18,476" and "18,674", this does not impact our comprehension that this column represents revenue). In contrast, the data type is critical for understanding spreadsheets. On one hand, data types represent fundamental semantic properties, such as "time" or "phone number". It motivates us to implement rules to match the value of the cell to different data types. On the other hand, in contrast to detailed numerical values, identical data types may be compressed through clustering, thereby reducing the number of tokens.

In this section, we introduce Data-format-aware Aggregation for further compression and information integration. Specifically, we employ Number Format String (NFS), which is a built-in cell attribute in spreadsheets. NFSs can be extracted by default using tools like ClosedXML or OpenPyXL, used to describe the format of cell data as a string. For instance, the NFS for "2024.2.14" is "yyyy-mm-dd", indicating a specific **date** format. However, spreadsheet users do not always explicitly add NFSs to cells, so NFSs are sometimes absent. As a complement, we propose a rule-based recognizer to map a cell value to a specific predefined data type: Year, Integer, Float, Percentage, Scientific notation, Date, Time, Currency, Email, and Others. The first nine types broadly cover approximately 55% of the cells in our dataset derived from real-world corpora. Finally, based on the NFSs and data type, the aggregator aggregates the cells by Algorithm 1. This process can be represented as follows:

$$\mathit{NFSs} = \text{nfs}(\{\mathit{Cell}_{i,j}\}_{i\in m, j\in n}), \quad (7)$$

$$\mathcal{T}_a = \text{aggregator}(\{\mathit{Cell}_{i,j}\}_{i\in m, j\in n}, \mathit{NFSs}, R), \quad (8)$$

where R denotes the predefined rules as detailed above. In this way, we further reduce the number of tokens. The compression ratio of the data regions also increases from 14.91 to 24.79. More detailed compression effects of different modules are displayed in Table 1.

3.5 Chain of Spreadsheet Encoding

To extend the applicability of SHEETENCODER to a broader range of downstream tasks, we introduce the Chain of Spreadsheet Encoding (CoS), which unfolds two stages:

Table Identification and Boundary Detection Initially, the compressed spreadsheet and the specific task query are input into the LLM. Leveraging the advances in spreadsheet table detection, the model identifies the table that is relevant to the query and determines the precise boundaries of the relevant content. This step ensures that only pertinent data is considered in the subsequent analysis, optimizing the processing efficiency and focus.

Response Generation The query and the identified table section are re-input detected tables into the LLM. The model then processes this information to generate an accurate response to the query.

Through the CoS, on one hand, spreadsheet downstream tasks can naturally be decoupled into spreadsheet understanding (via the table detection task) and further reasoning, advocating for a stepwise problem-solving mindset. On the other hand, in the further reasoning phase, CoS can be compatible with CoT (Wang et al.) for table reasoning by enabling LLMs to reason step by step. In this paper, we validate the effectiveness of the Spreadsheet QA task, which is detailed in Section 4.2.

4 Experiments

In our experimental evaluation, we first verified the effectiveness of our method in spreadsheet understanding. For this purpose, we chose the classic and foundational task of spreadsheet table detection (Dong et al., 2019b). This task serves as a critical benchmark for assessing the framework's ability to accurately identify and interpret table structures within spreadsheets. Building upon this foundational understanding, we further explored the applicability of our method to downstream applications by selecting the representative task of spreadsheet QA. This allows us to test the model's capability to not only detect but also comprehend and respond to user queries based on the data and structure identified in the spreadsheets.

4.1 Spreadsheet Table Detection

4.1.1 Dataset

We used the dataset introduced by (Dong et al., 2019b), a benchmark dataset of real-world spreadsheets with annotated table boundaries. Due to the complexity and ambiguity of precise address labeling (the Fleiss Kappa on the test set is 0.830), we further implemented the quality improvement pipeline on the test set by five human professions, as detailed n in Appendix E. To this end, we obtained a highly validated test set containing 188 spreadsheets. Based on the token usage of the vanilla encoding method, we divided the test set into four categories: Small, Medium, Large, and Huge, with a partition of 64:32:70:22. More details are shown in Appendix F. We adopted the Error-of-Boundary 0 (EoB-0) metric for evaluation on 188 spreadsheets with 311 tables. EoB-0 requires **exact match** of the top, left, bottom, and right boundaries.

4.1.2 Experiment Setup

Baseline & Evaluation Metrics To evaluate the performance of SHEETENCODER, we chose TableSense-CNN (Dong et al., 2019b) as the baseline due to its previously demonstrated effectiveness in spreadsheet table detection task. We employed the micro F1 Score as the primary metric to evaluate and compare the performance of different models, as it balances precision and recall, providing a holistic view of model accuracy.

Model Selection The experiments included both closed-source and open-source models. From the closed-source spectrum, we selected two versions of OpenAI's models: GPT4 and GPT3.5, which are known for their advanced language understanding capabilities. On the open-source side, we chose Llama2, Llama3, Phi3, and Mistral-v2. The specific configurations are detailed in Appendix G.

4.2 Spreadsheet QA

4.2.1 Dataset

Existing datasets for the Table QA task focus solely on single-table scenarios, leaving a notable gap in performance evaluation for spreadsheets that contain multiple tables. To bridge this gap, we developed a new Spreadsheet QA dataset tailored to the complexities of multi-table environments. We sampled 64 spreadsheets from our larger collection and crafted 4-6 questions per spreadsheet, targeting fundamental operations such as searching, comparison, and basic arithmetic. We deliberately excluded questions involving composite operations to maintain clarity and focus in testing specific skills. Each question was paired with an answer, formatted either as a specific cell address or a formula that includes cell addresses, facilitating direct and unambiguous evaluations of the model's ability to navigate and interpret spreadsheet data. This approach resulted in a comprehensive test dataset comprising 307 items, each a tuple of (Q, A, S), which is detailed in Appendix H.

4.2.2 Experiment Setup

Baseline & Evaluation Metrics Given that LLMs have not yet been systematically applied to Spreadsheet QA tasks, we have selected TAPEX and Binder (Liu et al.; Cheng et al.), which are established baselines in the Table QA domain, for comparative evaluation. Since TAPEX and Binder are designed primarily for single-table data, we adapted them for our multi-table context. Initially, our fine-tuned model identifies table regions relevant to each question. These regions are then formatted and fed into the baseline models. In cases where the input exceeds the token limitations of the baseline models, truncation is employed. The accuracy of the answers is assessed based on the correctness of the cell addresses and cell combinations/calculations provided in the answers.

Model Selection Our experiments were conducted using the GPT4 model, leveraging its advanced capabilities in language understanding and reasoning. Details on parameters and configurations used are documented in Appendix G.

4.2.3 Experiment Procedure

In this paper, we fine-tuned LLMs using the training set of table boundary detection, which includes both uncompressed and compressed encoding methods. In the QA experiments, we utilized

Metric	No Modules	Module 1	Module 2	Module 3	Module 1&2	Module 1&3	Module 2&3	Module 1&2&3
Total Tokens	1,548,577	350,946	580,912	213,890	103,880	96,365	211,445	62,469
Compression Ratio	1.00	4.41	2.67	7.24	14.91	16.07	7.32	24.79

Table 1: Average Compression Ratio on test datasets. Results of the train & valid set are shown in Appendix J.1.

the fine-tuned model on the table boundary detection task for spreadsheet QA as an ablation to study the generalization capability of the fine- tuned boundary detection model. CoS supports multi-step reasoning during the response generation process. In the QA scenarios, the whole steps include spreadsheet table detection, table understanding, table splitting, and sub-table QA. In cases where tables were exceptionally large and defy effective compression, we utilized a table-splitting algorithm designed to recognize headers and perform strategic concatenation, ensuring that each segment of the split table retains as much contextual integrity as possible. The specifics of this algorithm are detailed in Appendix M.2. We categorically classify the model's responses as either correct or incorrect, and we select accuracy as the evaluation metric.

5 Results

5.1 Compression Ratio

The effectiveness of our encoding process in reducing the size of spreadsheet data is quantitatively assessed using the compression ratio, which is defined by the formula:

$$r = n/n', \tag{9}$$

Our encoding methodology has significantly optimized token usage within spreadsheets. In our test set, it achieved an impressive 25× compression ratio, substantially reducing the computational load for processing large datasets. The specific compression ratios achieved by various module combinations within SheetEncoder are detailed in Table 1. These results highlight the efficacy of our approach across different configurations, demonstrating its robustness and adaptability in handling diverse spreadsheet structures.

5.2 Spreadsheet Table Detection

5.2.1 Main Results

Table 2 illustrates the performance differences among various models and methods on spreadsheet table detection task, and the detailed case study can refer to Appendix K.

Model & Method	Small	Medium	Large	Huge	All
ICL					
Mistral-v2	0.071	0.013	0.029	0.017	0.036
GPT4	0.318	0.292	0.090	0.000	0.154
GPT4-compress	0.480	0.454	0.373	0.330	0.410
Fine-tune					
Llama3	0.715	0.765	0.290	0.000	0.471
Llama2	0.557	0.378	0.107	0.000	0.280
Phi3	0.604	0.481	0.201	0.130	0.330
Mistral-v2	0.700	0.784	0.472	0.123	0.542
GPT4	0.779	0.707	0.288	0.000	0.520
Llama3-compress	0.825	0.768	0.664	0.617	0.719
Llama2-compress	0.710	0.722	0.633	0.578	0.660
Phi3-compress	0.800	0.673	0.624	0.675	0.689
Mistral-v2-compress	0.778	0.729	0.686	0.744	0.726
GPT3.5-compress	0.795	0.649	0.600	0.680	0.680
GPT4-compress	0.810	**0.832**	0.718	0.690	0.759
-w/o Aggregation	**0.864**	0.816	**0.739**	**0.753**	**0.789**
TableSense-CNN	0.785	0.788	0.567	0.561	0.666

Table 2: Results of various Model & Method configurations on spreadsheet table detection in the EOB-0 metric. "-compress" indicates the usage of compression modules in SheetEncoder. GPT4 with SheetEncoder achieved the best results.

1) Enhanced Performance with various LLMs: The fine-tuned GPT4 model achieved the F1 score of approximately 76% across all datasets, while our encoding method without aggregation achieved the F1 score of approximately 79% across all datasets. This marked a 27% percentage points improvement over the same model fine-tuned on original data, a 13% percentage points increase over TableSense-CNN. The entire encoding method slightly reduced the F1 score within a tolerable range, but achieved good compression results, as shown in Table 1. We also evaluated our encoding method on a series of open-source models. Notably, Llama3 and Mistral-v2 achieved an F1 score of approximately 72%, just 6 percentage points below the best method. The improvements due to our compression method were substantial, with increases of 25% for Llama3, 36% for Phi3, 38% for Llama2, and 18% for Mistral-v2. These results underscored the significant enhancement performance attributable to our encoding method.

2) Benefits for Larger Spreadsheets: Our com-

Model	Small	Medium	Large	Huge	All
GPT4	0.779	0.700	0.288	0.000	0.520
GPT4-compress	0.810	0.832	0.718	0.690	0.759
-w/o Extraction	0.805	0.772	0.618	0.321	0.655
-w/o Translation	0.785	0.804	0.729	0.636	0.743
-w/o Aggregation	0.864	0.816	0.739	0.753	0.789

Table 3: Ablation studies on spreadsheet table detection.

pression method significantly boosted performance on larger spreadsheets, where the challenges were most pronounced due to model token limits. The improvements in F1 scores were particularly notable on huge spreadsheets (75% over GPT4, 19% over TableSense-CNN), large spreadsheets (45% and 17%), medium (13% and 5%), and small (8%) spreadsheets. This demonstrated our method's effectiveness in enabling LLMs to process a broader range of spreadsheet sizes efficiently.

3) Improvements in In-Context Learning: Compact encoding also significantly enhanced ICL capabilities. For instance, the performance of GPT4 on all data improved by nearly 26%, demonstrating the method's effectiveness beyond fine-tuned models to include ICL scenarios as well. More ICL results are shown in Appendix J.2.

4) Significant Cost Reduction: Our cost was almost directly proportional to the input tokens, because the output table regions are short, which can be neglected. Based on the prices of the GPT4 and GPT3.5-turbo models [1] in ICL, we reduced 96% cost in our test set. Detailed calculations are presented in Appendix I.

5.2.2 Ablation Experiment Results

Table 3 presents the results of ablation experiments for different modules. The removal of the extraction module led to significantly lower F1 scores, underscoring its critical role in capturing and retaining key structural information. As highlighted in Table 1, this module also achieved the most significant token reduction, confirming its effectiveness. After removing the aggregation module, the F1 score slightly increased. This observation might be attributed to the NFS being more abstract than straightforward numerical representations, which can challenge an LLMs' ability to interpret them effectively. Despite this, the NFS method offered a significantly high compression rate, enhancing its potential for practical applications and cost control.

[1]https://azure.microsoft.com/en-us/pricing/details/cognitive-services/openai-service/

Model	Accuracy
TAPEX	0.378
Binder	0.622
GPT4	0.466
GPT4-compress-w/o splitting	0.651
GPT4-compress-w/o splitting-FT	0.694
GPT4-compress	0.684
GPT4-compress-FT	**0.743**

Table 4: The results for Spreadsheet QA show that our method achieved highest accuracy. "-FT" means fine-tuned model on spreadsheet table detection task and is applied to QA.

5.3 Spreadsheet QA

Table 4 shows the accuracy of various models on Spreadsheet QA tasks. We can draw the following conclusions:

1) Effectiveness of the CoS Method: The CoS method with SHEETENCODER significantly boosted the accuracy of models, showing a notable increase of 22% over the baseline GPT4 model. Given the large size of typical spreadsheets, directly inputting entire files often exceeded the token limits of conventional models. Some large tables even exceed the token limitation after compression. The CoS effectively addressed this issue by focusing only on regions relevant to the questions posed, improving accuracy by 3% and 5% on ICL and fine-tuning respectively.

In structural-anchor-based extraction, some information loss may occur, which could potentially affect downstream tasks. The CoS method was proposed to address this issue. Based on the overall understanding of the spreadsheet and the table detection, CoS can identify the related table and re-encode it for downstream tasks without structural-anchor-based extraction. Additionally, the three modules proposed in SHEETENCODER are modular and can be freely chosen and used according to different tasks. For instance, the invert-index translation module does not result in any information loss and is almost applicable to all downstream tasks.

In the works of TAPEX and Binder that we referenced, the tables are unlike those in spreadsheets, which can feature a multi-table layout, hundreds of rows (or even more), and formats such as merged cells that complicate the understanding of spatial information. To mitigate these challenges and enhance their capabilities on QA tasks, we have used tables detected in the spreadsheet that were related to the query as inputs. On the other hand, the address information of large tables in spreadsheets is crucial. In SHEETENCODER, we have augmented each cell with coordinate information. We speculate that if TAPEX's pre-training had considered this setting, its performance might have further improved.

2) Generalization Capability of the Fine-tuned Model: The model that has been fine-tuned on the spreadsheet table detection task demonstrated robust generalization capabilities across downstream QA tasks. This fine-tuning on table detection led to an accuracy improvement of 6% on QA. Moreover, it significantly outperformed the TAPEX and Binder models by 37% and 12%, respectively. This substantial margin highlighted that fine-tuning not only prepared the model to better understand the specific data and structural nuances of spreadsheets but also enhanced its overall comprehension abilities.

6 Conclusion

In this paper, we proposed the SHEETENCODER, a novel framework representing a significant advancement in the processing and understanding of spreadsheet data by leveraging the capabilities of LLMs. Through a novel efficient encoding method, this framework effectively addresses the challenges posed by the size, diversity, and complexity inherent in spreadsheets. It achieves a substantial reduction in token usage and computational costs, enabling practical applications on large datasets. The fine-tuning of various cutting-edge LLMs further enhances the performance of spreadsheet understanding. Moreover, Chain of Spreadsheet Encoding, the framework's extension to spreadsheet downstream tasks illustrates its broad applicability and potential to bridge spreadsheet data management and analysis with table-based techniques, paving the way for more intelligent and efficient user interactions.

Limitations

While SHEETENCODER has markedly advanced how LLMs interpret and utilize spreadsheets, they also illuminate areas ripe for further research and development. Currently, our methods do not yet harness spreadsheet format details such as background color and borders, because they take too many tokens. However, these elements often contain valuable contextual and visual cues that could further refine our understanding and processing of spreadsheet data. Additionally, while SHEETENCODER effectively aggregates data regions, it does not currently employ a sophisticated semantic-based compression method for cells containing natural language. For example, categorizing terms like "China," "America," and "France" under a unified label such as "Country" could not only increase the compression ratio but also deepen the semantic understanding of the data by LLMs. Exploring these advanced semantic compression techniques will be a key focus of our ongoing efforts to enhance the capabilities of SHEETENCODER.

Ethics Statement

All data were collected, analyzed, and reported without any bias or influence from external sources. The privacy and confidentiality of the participants were strictly maintained throughout the research process. No personal identifiers were used in the analysis or reporting of the data to ensure anonymity. At the same time, data standard personnel were paid according to the highest local standard, and their daily working hours were strictly limited to no more than 8 hours to protect their legitimate rights and interests. We acknowledge the contributions of all individuals and institutions involved in this study and are committed to sharing our findings and methodologies transparently to facilitate further research and knowledge advancement in the field.

References

Sibei Chen, Yeye He, Weiwei Cui, Ju Fan, Song Ge, Haidong Zhang, Dongmei Zhang, and Surajit Chaudhuri. 2024. Auto-formula: Recommend formulas in spreadsheets using contrastive learning for table representations. *Proceedings of the ACM on Management of Data*, 2(3):1–27.

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

Zhe Chen and Michael Cafarella. 2013. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search over the Web*, pages 1–8.

Zhe Chen and Michael Cafarella. 2014. Integrating spreadsheet data via accurate and low-effort extraction. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1126–1135.

Zhoujun Cheng, Haoyu Dong, Ran Jia, Pengfei Wu, Shi Han, Fan Cheng, and Dongmei Zhang. 2022a. Fortap: Using formulas for numerical-reasoning-aware table pretraining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1150–1166.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022b. 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)*, pages 1094–1110.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. Binding language models in symbolic languages. In *The Eleventh International Conference on Learning Representations*.

Christina Christodoulakis, Eric B Munson, Moshe Gabel, Angela Demke Brown, and Renée J Miller. 2020. Pytheas: pattern-based table discovery in csv files. *Proceedings of the VLDB Endowment*, 13(12):2075–2089.

Naihao Deng, Zhenjie Sun, Ruiqi He, Aman Sikka, Yulong Chen, Lin Ma, Yue Zhang, and Rada Mihalcea. 2024. Tables as images? exploring the strengths and limitations of llms on multimodal representations of tabular data. *arXiv preprint arXiv:2402.12424*.

Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record*, 51(1):33–40.

Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. 2022. Table Pretraining: A survey on model architectures, pretraining objectives, and downstream tasks. *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*.

Haoyu Dong, Shijie Liu, Zhouyu Fu, Shi Han, and Dongmei Zhang. 2019a. Semantic structure extraction for spreadsheet tables with a multi-task learning architecture. In *Workshop on Document Intelligence at NeurIPS 2019*.

Haoyu Dong, Shijie Liu, Shi Han, Zhouyu Fu, and Dongmei Zhang. 2019b. TableSense: Spreadsheet table detection with convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 69–76.

Haoyu Dong and Zhiruo Wang. 2024. Large language models for tabular data: Progresses and future directions. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2997–3000.

Wensheng Dou, Shi Han, Liang Xu, Dongmei Zhang, and Jun Wei. 2018. Expandable group identification in spreadsheets. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 498–508.

Wensheng Dou, Chang Xu, Shing-Chi Cheung, and Jun Wei. 2016. Cacheck: detecting and repairing cell arrays in spreadsheets. *IEEE Transactions on Software Engineering*, 43(3):226–251.

Iyad Abu Doush and Enrico Pontelli. 2010. Detecting and recognizing tables in spreadsheets. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 471–478.

Majid Ghasemi Gol, Jay Pujara, and Pedro Szekely. 2019. Tabular cell classification using pre-trained cell embeddings. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 230–239. IEEE.

Xinyi He, Mengyu Zhou, Xinrun Xu, Xiaojun Ma, Rui Ding, Lun Du, Yan Gao, Ran Jia, Xu Chen, Shi Han, et al. 2024. Text2analysis: A benchmark of table question answering with advanced data analysis and unclear queries. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18206–18215.

Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1500–1508.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023a. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023b. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. Omnitab: Pretraining with natural and synthetic data for few-shot table-based question answering. *arXiv preprint arXiv:2207.03637*.

Harshit Joshi, Abishai Ebenezer, José Cambronero Sanchez, Sumit Gulwani, Aditya Kanade, Vu Le, Ivan Radiček, and Gust Verbruggen. 2024. Flame: A small language model for spreadsheet formulas. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 12995–13003.

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

Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chaudhuri. 2024. Auto-tables: Relationalize tables without using examples. *ACM SIGMOD Record*, 53(1):76–85.

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

Yucheng Li. 2023. Unlocking context constraints of llms: Enhancing context efficiency of llms with self-information-based content filtering. *arXiv preprint arXiv:2304.12102*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. Tapex: Table pre-training via learning a neural sql executor. In *International Conference on Learning Representations*.

Kyosuke Nishida, Kugatsu Sadamitsu, Ryuichiro Higashinaka, and Yoshihiro Matsuo. 2017. Understanding the semantic structures of tables with a hybrid deep neural network architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2023. Gpt4table: Can large language models understand structured table data? a benchmark and empirical study. *arXiv preprint ArXiv:2305.13062*.

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*, pages 645–654.

Gerardo Vitagliano, Lucas Reisener, Lan Jiang, Mazhar Hameed, and Felix Naumann. 2022. Mondrian: Spreadsheet layout detection. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2361–2364.

Pei Wang and Yeye He. 2019. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the 2019 International Conference on Management of Data*, pages 811–828.

Tianshu Wang, Hongyu Lin, Xianpei Han, Le Sun, Xiaoyang Chen, Hao Wang, and Zhenyu Zeng. 2023. Dbcopilot: Scaling natural language querying to massive databases. *arXiv preprint arXiv:2312.03463*.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. 2021. TUTA: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*.

Shiyu Xia, Junyu Xiong, Haoyu Dong, Jianbo Zhao, Yuzhang Tian, Mengyu Zhou, Yeye He, Shi Han, and Dongmei Zhang. 2024. Vision language models for spreadsheet understanding: Challenges and opportunities. *arXiv preprint arXiv:2405.16234*.

Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Retrieval meets long context large language models. *arXiv preprint arXiv:2310.03025*.

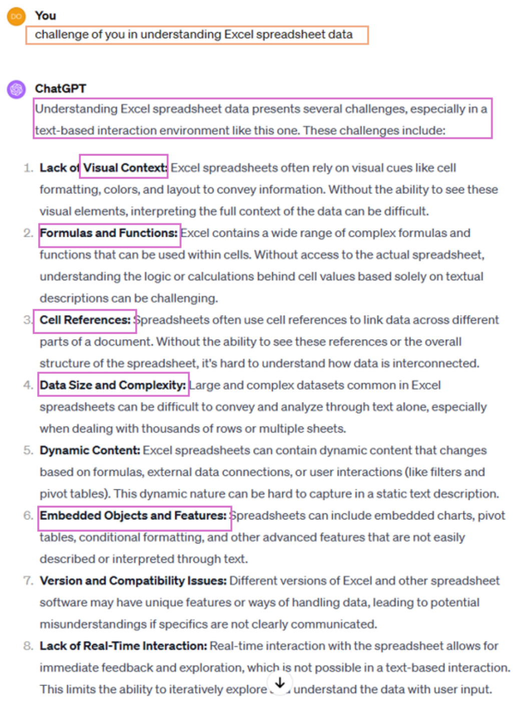
Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2024a. Tablellama: Towards open large generalist models for tables. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6024–6044.

Xiaokang Zhang, Jing Zhang, Zeyao Ma, Yang Li, Bohan Zhang, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-Li, et al. 2024b. Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios. *arXiv preprint arXiv:2403.19318*.

Mingyu Zheng, Hao Yang, Wenbin Jiang, Zheng Lin, Yajuan Lyu, Qiaoqiao She, and Weiping Wang. 2023. Chain-of-thought reasoning in tabular language models. pages 11006–11019.

A GPT4 Struggles to Understand Spreadsheets

The Figure 3 and Figure 4 show how GPT4 struggles to understand spreadsheets. We also validated the effect of cell format on the vanilla encoding method on the spreadsheet table detection task. As shown in Table 5, the results indicate that in ICL, the inclusion of format marginally improves the model's performance on small datasets but results in poorer performance on larger datasets. For the fine-tuned model, the inclusion of format information leads to a significant reduction in the overall F1 score. This decline is attributed to the introduction of additional tokens, which causes some data to exceed the model's token limits. Additionally, LLMs are not yet adept at understanding format information.



You
challenge of you in understanding Excel spreadsheet data

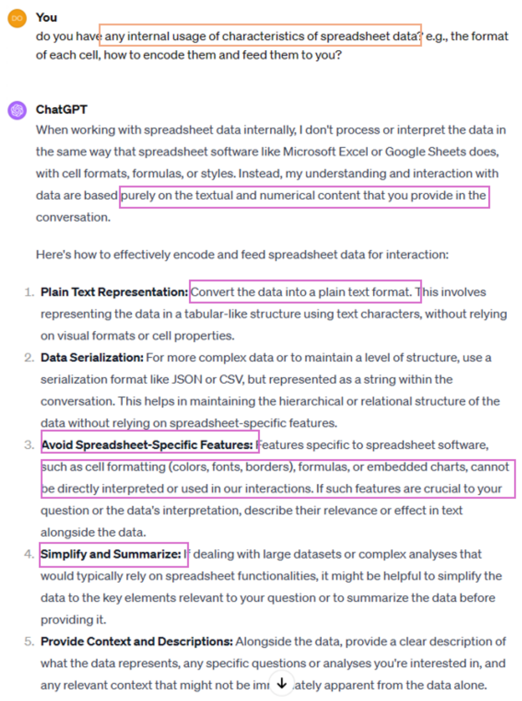
ChatGPT
Understanding Excel spreadsheet data presents several challenges, especially in a text-based interaction environment like this one. These challenges include:

1. **Lack of Visual Context:** Excel spreadsheets often rely on visual cues like cell formatting, colors, and layout to convey information. Without the ability to see these visual elements, interpreting the full context of the data can be difficult.
2. **Formulas and Functions:** Excel contains a wide range of complex formulas and functions that can be used within cells. Without access to the actual spreadsheet, understanding the logic or calculations behind cell values based solely on textual descriptions can be challenging.
3. **Cell References:** Spreadsheets often use cell references to link data across different parts of a document. Without the ability to see these references or the overall structure of the spreadsheet, it's hard to understand how data is interconnected.
4. **Data Size and Complexity:** Large and complex datasets common in Excel spreadsheets can be difficult to convey and analyze through text alone, especially when dealing with thousands of rows or multiple sheets.
5. **Dynamic Content:** Excel spreadsheets can contain dynamic content that changes based on formulas, external data connections, or user interactions (like filters and pivot tables). This dynamic nature can be hard to capture in a static text description.
6. **Embedded Objects and Features:** Spreadsheets can include embedded charts, pivot tables, conditional formatting, and other advanced features that are not easily described or interpreted through text.
7. **Version and Compatibility Issues:** Different versions of Excel and other spreadsheet software may have unique features or ways of handling data, leading to potential misunderstandings if specifics are not clearly communicated.
8. **Lack of Real-Time Interaction:** Real-time interaction with the spreadsheet allows for immediate feedback and exploration, which is not possible in a text-based interaction. This limits the ability to iteratively explore understand the data with user input.

Figure 3: Challenges of GPT4 understanding spreadsheet data.

Model	Small	Medium	Large	Huge	All
GPT4-ICL	0.318	0.292	0.090	0.000	0.154
GPT4-ICL-FMT	0.429	0.000	0.000	0.000	0.204
GPT4-FT	0.779	0.707	0.288	0.000	0.520
GPT4-FT-FMT	0.758	0.000	0.000	0.000	0.315

Table 5: The results of spreadsheet table detection experiment with cell format.



You
do you have any internal usage of characteristics of spreadsheet data? e.g., the format of each cell, how to encode them and feed them to you?

ChatGPT
When working with spreadsheet data internally, I don't process or interpret the data in the same way that spreadsheet software like Microsoft Excel or Google Sheets does, with cell formats, formulas, or styles. Instead, my understanding and interaction with data are based purely on the textual and numerical content that you provide in the conversation.

Here's how to effectively encode and feed spreadsheet data for interaction:

1. **Plain Text Representation:** Convert the data into a plain text format. This involves representing the data in a tabular-like structure using text characters, without relying on visual formats or cell properties.
2. **Data Serialization:** For more complex data or to maintain a level of structure, use a serialization format like JSON or CSV, but represented as a string within the conversation. This helps in maintaining the hierarchical or relational structure of the data without relying on spreadsheet-specific features.
3. **Avoid Spreadsheet-Specific Features:** Features specific to spreadsheet software, such as cell formatting (colors, fonts, borders), formulas, or embedded charts, cannot be directly interpreted or used in our interactions. If such features are crucial to your question or the data's interpretation, describe their relevance or effect in text alongside the data.
4. **Simplify and Summarize:** If dealing with large datasets or complex analyses that would typically rely on spreadsheet functionalities, it might be helpful to simplify the data to the key elements relevant to your question or to summarize the data before providing it.
5. **Provide Context and Descriptions:** Alongside the data, provide a clear description of what the data represents, any specific questions or analyses you're interested in, and any relevant context that might not be imr ately apparent from the data alone.

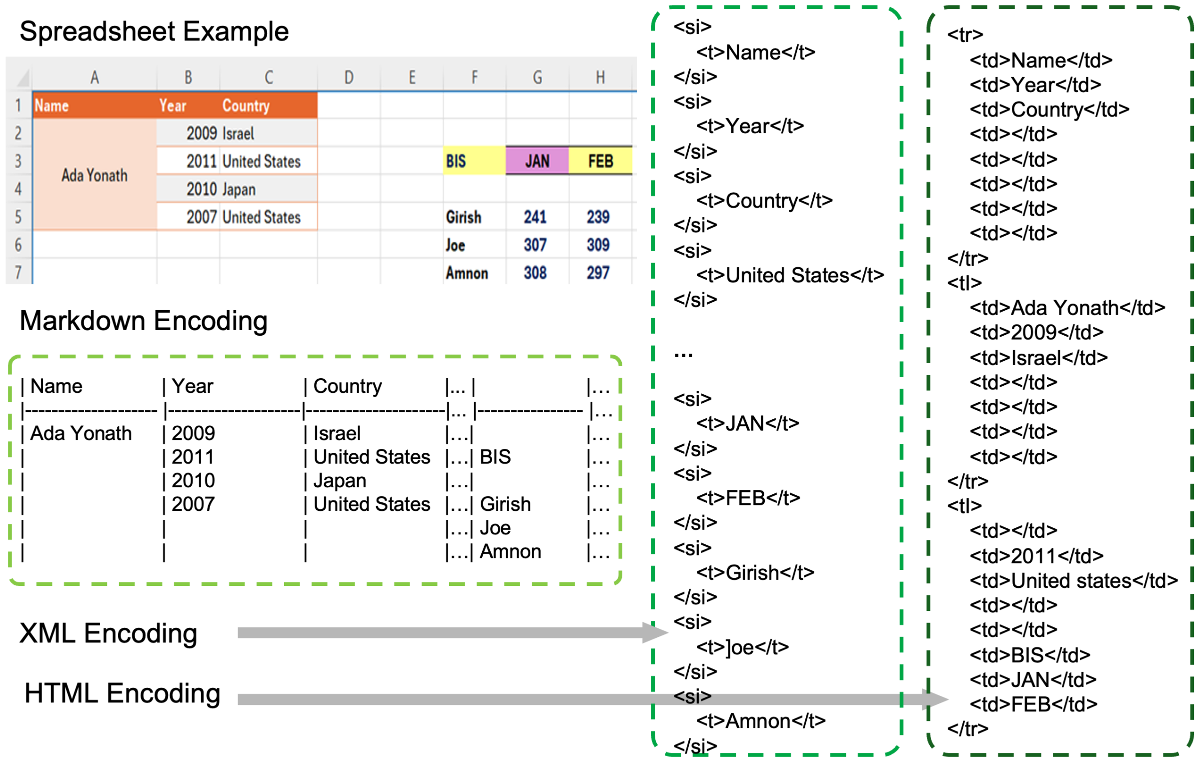
Figure 4: GPT4 encoding methods and techniques for processing spreadsheet data.

B Traditional Encoding Methods for Spreadsheets

In our study, we explored traditional encoding methods—Markdown, XML, and HTML—to represent spreadsheet data. Figure 5 illustrates the comparative analysis of these methods. XML and HTML encoding, while widely used, tend to result in high token consumption due to the extensive use of repeated label tags necessary for representing the data structure. This approach markedly increases the volume of data processed.

Conversely, the Markdown method, although more token-efficient, has its limitations. One significant drawback is the lack of explicit cell address information, which frequently leads to errors when indexing specific cell locations. Additionally, Markdown's rigid structure rules complicate the accurate representation of merged cells, a common feature in complex spreadsheets that is crucial for preserving the integrity of data relationships.

To quantitatively assess these methods, we conducted ICL experiments using the GPT4 model on spreadsheet detection tasks. The results, detailed in Table 6, confirmed that while the Markdown method outperformed XML and HTML in terms of lower token usage, it still fell short in addressing the needs of spreadsheet encoding effectively.



	A	B	C	D	E	F	G	H
1	Name	Year	Country					
2	Ada Yonath	2009	Israel					
3		2011	United States			BIS	JAN	FEB
4		2010	Japan					
5		2007	United States			Girish	241	239
6						Joe	307	309
7						Amnon	308	297

```
| Name        | Year        | Country        |... |            |...
|-------------------- |--------------------|--------------------|... |---------------- |...
| Ada Yonath  | 2009        | Israel         |...|             |...
|             | 2011        | United States  |...| BIS         |...
|             | 2010        | Japan          |...|             |...
|             | 2007        | United States  |...| Girish      |...
|             |             |                |...| Joe         |...
|             |             |                |...| Amnon       |...
```

```
<si>
    <t>Name</t>
</si>
<si>
    <t>Year</t>
</si>
<si>
    <t>Country</t>
</si>
<si>
    <t>United States</t>
</si>

...

<si>
    <t>JAN</t>
</si>
<si>
    <t>FEB</t>
</si>
<si>
    <t>Girish</t>
</si>
<si>
    <t>]oe</t>
</si>
<si>
    <t>Amnon</t>
</si>
```

```
<tr>
    <td>Name</td>
    <td>Year</td>
    <td>Country</td>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
</tr>
<tl>
    <td>Ada Yonath</td>
    <td>2009</td>
    <td>Israel</td>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
</tr>
<tl>
    <td></td>
    <td>2011</td>
    <td>United states</td>
    <td></td>
    <td></td>
    <td>BIS</td>
    <td>JAN</td>
    <td>FEB</td>
</tr>
```

Figure 5: Examples of three traditional spreadsheet encoding methods: Markdown, XML, and HTML. Due to space limitations, we only show the encoding of some cells.

	Small	Medium	Large	Huge	All
HTML	0.074	0.016	0.003	0.000	0.031
XML	0.292	0.102	0.066	0.000	0.142
Markdown	0.254	0.167	0.143	0.121	0.175

Table 6: The ICL experiments of different encoding methods of the spreadsheet on GPT4.

C Lightweight Heuristics for Structural-anchor Proposal

Initially, this method enumerates bounding lines by finding discrepancies in neighboring rows and columns based on differences in cell values, merged cells, borders, and fill color. In other words, it enumerates rows and columns with imbalances (text, merge, border, color, font, etc.). Rows and columns without significant discrepancies are usually canonical data rows or columns that contribute trivially to the layout understanding of a spreadsheet. Subsequently, it composes all possible candidate boundaries using any two rows and any two columns as top/bottom/left/right edges. In the third step, heuristics are applied to filter unreasonable boundary candidates by judging the integrity within each candidate boundary. For example, the proportion of numbers and characters in each row and column is used to infer the sparsity in the internal region and four edges of the candidate boundary. The size of the boundary is used to infer if it is too small to be a table, and the presence of header-like rows and columns is also considered. After this step, a small proportion of candidate boundaries are preserved.

In the fourth step, overlapping candidate boundaries are enumerated pairwise. Information such as the relative positions of candidate boundaries and the presence of headers is used to determine which candidate boundary more likely represents a table, thereby filtering out some overlapping boundaries. Figure 6 presents common overlapping patterns. For example, for two overlapping candidate boundaries with close top boundaries, heuristics use the proportion of textual cells or format strings like year and date to determine candidate headers. The existence of candidate headers is then used to decide which candidate boundary to filter out.

Finally, we take the remaining candidate boundaries to derive structural anchors. However, due to the challenge of fine-grained discriminating headers, titles, and notes for heuristics, the candidate boundaries produced by the above heuristics only achieve 46.3% F1 score in EoB-0 metric and 65.0% EoB-2 metric in our boundary detection test set. Fortunately, including neighboring rows and columns largely increases the coverage of real bounding rows and columns, because headers, titles, and notes usually span few rows and columns. So we propose to not only use the exact bounding rows and columns as structural anchors but also include rows and columns within k rows and columns neighboring the structural anchors to preserve titles, notes, and headers as much as possible. This allows LLMs to further determine the exact boundaries by leveraging their advanced capabilities in semantic understanding and reasoning. When k is set to 4, over 97% of the border rows and columns in ground truth tables are preserved. This ensures that structure anchors rarely lose critical information of the table skeleton.

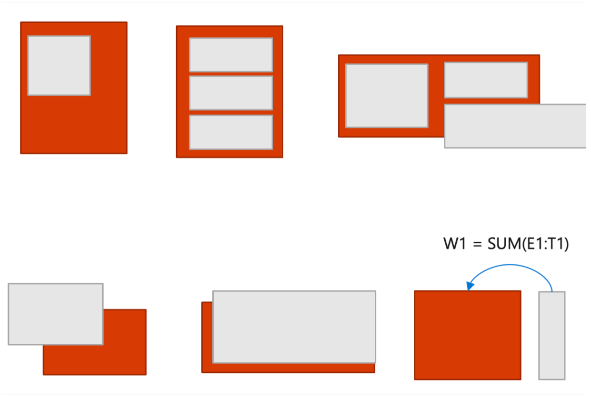


Figure 6: Common overlapping patterns of candidate boundaries.

D Ablation Experiment Results of Spreadsheet Table Detection

D.1 Results on Structure-anchor Threshold

Table 7 details the ablation study concerning the number of rows and columns retained near candidate boundaries. Optimal results were observed when four rows/columns were preserved, yielding the highest F1 score across all datasets. This outcome is likely due to a balance between preserving essential boundary information and maintaining a feasible compression ratio. Retaining fewer rows/columns might omit critical boundaries, reducing Recall, while preserving more rows/columns diminishes the compression ratio, potentially exceeding the model's token limits.

For smaller data, results indicate a positive correlation between the number of retained rows and the F1 score, suggesting that higher information retention leads to better model performance.

k	Small	Medium	Large	Huge	All
2	0.775	0.804	0.686	0.558	0.712
4	0.810	0.832	0.718	0.690	0.759
8	0.788	0.824	0.773	0.400	0.744

Table 7: Spreadsheet table detection Ablation on extracted threshold k. We present experiment results of three different k: 2, 4, and 8 on fine-tuned GPT4.

Model	Small	Medium	Large	Huge	All
GPT4-compress	0.480	0.454	0.373	0.330	0.410
-w/o Aggregation	0.386	0.271	0.215	0.267	0.280
-w/o Translation	0.386	0.427	0.338	0.418	0.379
-w/o Extraction	0.345	0.263	0.198	0.268	0.257

Table 8: Ablation experiment results on ICL on spreadsheet table detection. Our compression method also achieved the best F1 score on ICL.

D.2 Results of Spreadsheet Table Detection on ICL

We conducted experiments on the GPT4, "GPT4-0125-preview" version. As shown in Table 8, the results are consistent with the conclusions we draw from our fine-tuned experiments.

E Spreadsheet Table Detection Test Dataset Quality Improvement Pipeline

The quality improvement pipeline on the test set consists of the following steps: (1) excluding those spreadsheets where at least one cell contains languages beyond English; (2) removing spreadsheets in the test set that lie in the same workbook as at least one spreadsheet in the training set, because spreadsheets in the same workbook, though different, are often similar; (3) annotating all spreadsheets in three types: type 1 means certain for one label; type 2 means multiple labels are reasonable; type 3 means not certain. We employ five well-educated annotators from top universities with majors in computer science to undertake this quality improvement. For each spreadsheet in the test set, we aggregate the annotations from all five annotators and preserve multiple reasonable labeling results for type 2 spreadsheets.

As a result, we obtained a well-annotated dataset with 167 spreadsheets containing 268 tables for type 1, 21 spreadsheets with 43 tables for type 2, and 10 spreadsheets for type 3. We selected data labeled as type 1 and type 2 as the test set, comprising a total of 188 entries.

F Spreadsheet Table Detection Test Dataset Partition

From the spreadsheet raw file, we can extract various features, including cell address, value, format (background color, bold, borders, etc.), and more. We transformed these features into the markdown-like style in Section3.1. Then, based on the number of tokens after encoding and the length of the context window of the test model, we divided them into four categories: small (number of tokens less than 4k), medium (4-8k), large (8-32k), and huge (greater than 32k). The following is an example of data in Markdown with format information.

Example: Encoding Spreadsheet in Markdown-like Style with Cell Formats

```
Text Input:
B2,Table 4: Diesel-driven passenger cars, 2015|C2, |D2, |E2, |F2, |G2, |H2,
B3, |C3, |D3, |E3, |F3, |G3, |H3,
B4, |C4, |D4, |E4, |F4, |G4, |H4,
|B5, |C5,Diesel engine|D5, |E5, |F5,Share of all passenger cars (%)|G5, |H5,
......
Format Input:
|B2,Font Bold|C2, |D2, |E2, |F2, |G2, |H2,
|B3, |C3, |D3, |E3, |F3, |G3, |H3,
|B4,Bottom Border,|C4,Bottom Border,|D4,Bottom Border,|E4,Bottom Border,|F4,Bottom Border,|G4,Bottom Border,|H4,Bottom Border,
|B5,Top Border,Right Border,Fill Color,Font Bold|C5,Top Border,Bottom Border,Left Border,Fill Color,Font Bold|D5,Top Border,Bottom Border,Fill Color,Font Bold|E5,Top Border,Bottom Border,Right Border,Fill Color,Font Bold|F5,Top Border,Bottom Border,Left Border,Fill Color,Font Bold|G5,Top Border,Bottom Border,Fill Color,Font Bold|H5,Top Border,Bottom Border,Fill Color,Font Bold
......
```

G Experiment Setup

Open-source model using Deepspeed for distributed training on a workstation with 8 A100 GPUs by LoRA.

Llama2:meta-Llama/Llama-2-7b-chat-hf;

Llama3:meta-Llama/Meta-Llama-3-8B-Instruct;

Mistral-v2:mistralai/Mistral-v2-7B-Instruct-v0.2;

Phi3:microsoft/Phi-3-mini-128k-instruct;

The parameters of open-source model fine-tuning: cutoff len=5800; learning rate=5e-05; num train epochs=15.0; train batch size=5; gradient accumulation steps=8; lr scheduler type is cosine; max grad norm=1.0; warmup steps=0; optim is AdamW; precision is fp16; lora rank=32; lora alpha=64; lora dropout=0.01; val size=0.0008; eval steps=50; eval batch size=5

The parameters of GPT4/3.5 model fine-tuning: lora_dim=32,n_epochs=-1,batch_size=-1,learning_rate_multiplier=1,weight_decay_multiplier=1e-05,prompt_loss_weight=0,trim_mode=right.

The parameters of GPT4/3.5 model inference: temperature=0, max tokens=300, top p=0.95, frequency penalty=0, presence penalty= 0, stop=None, and the rest are default settings.

H Spreadsheet QA Test Dataset

Overall Description The dataset of 64 spreadsheets includes 9 single table spreadsheets, 35 double table spreadsheets, 11 spreadsheets containing three tables, and 9 spreadsheets containing four or more tables. Among them, 15 spreadsheets contain fewer than 4k tokens, 20 contain between 4k and 8k, 22 contain between 8k and 32k, and 7 contain more than 32k.

Details of the Dataset Collection We selected English-language spreadsheets and invited five well-educated professional annotators to annotate the data. During selection, spreadsheets containing non-ASCII characters or lacking necessary semantic comprehension information were excluded. We ensured that the questions could be answered with relative certainty using the information provided in the tables, minimizing the potential confusion or ambiguity. To further validate the quality of the dataset, we invited two additional annotators to perform cross-verification after the initial question-answer labeling process, ensuring the correctness and rationality of the answers. It shows an answer accuracy of 0.846 in Fleiss Kappa, indicating almost perfect agreement.

Example: Spreadsheet QA Data Item

QUESTION: "What were the highest temperatures in Washington DC in 1998?"
GROUNDTRUTH: "X23 AND X24"
PROMPT: [Instruction + Encoded Spreadsheet]

I Cost calculation

We use the ICL price of GPT4 due to the absence of fine-tuned GPT4's price. We neglect the output sequence since it is much shorter than the input sequence in tasks like spreadsheet boundary detection and QA. The average cost of processing a spreadsheet in our test set has decreased by 96.0% for GPT3.5 turbo and GPT4, saving an impressive 96.0% in costs. The cost reduction similarly applies to all LLMs we used.

J Other Experimental Results

J.1 Compression Results

Table 9 shows the compression ratio of each stage in our method relative to the previous stage.

Table 10 shows the total compression ratio of train and valid datasets.

J.2 The ICL results of open-source models on spreadsheet table detection.

Table 11 shows the ICL experiments' F1 score of open-source models on the spreadsheet table detection task. In this experimental setting, the open-source model performs far worse than the closed-source model.

J.3 Spreadsheet QA Ablation Experiment

Table 12 assesses the impact of removing individual modules on the QA performance. It details both the overall accuracy and the accuracy of identifying question-related regions during the CoT process. The removal of any module generally leads to a decrease in both metrics, with the most significant drop observed when the extraction module is omitted. This is likely due to the extraction module achieving the lowest compression ratio (see Table 1), suggesting that a longer context may hinder the model's ability to accurately interpret the data.

Metric	No Modules	Module 1	Module 1&2	Module 1&2&3
Total Tokens	1,548,577	350,946	103,880	62,469
Compression Ratio	1.00	4.41	3.38	1.66

Table 9: Compression Ratio of Data Region.

Metric	Before	After
Valid Datasets (200 items)		
Tokens	1,462,076	99,411
Compression Ratio	1.00	14.71
Train Datasets (7000 items)		
Tokens	192,879,819	11,392,870
Compress Ratio	1.00	16.93

Table 10: Compression performance on train & valid Datasets.

Model	Small	Medium	Large	Huge	All
Llama3	0.042	0.028	0.020	0.018	0.027
Llama2	0.062	0.023	0.038	0.027	0.041
Phi3	0.037	0.040	0.041	0.000	0.034
Mistral-v2	0.071	0.013	0.029	0.017	0.036

Table 11: The ICL results of open-source models' performance on spreadsheet table detection.

K Case Study

K.1 Comparison of results before and after structural-anchor-based extraction

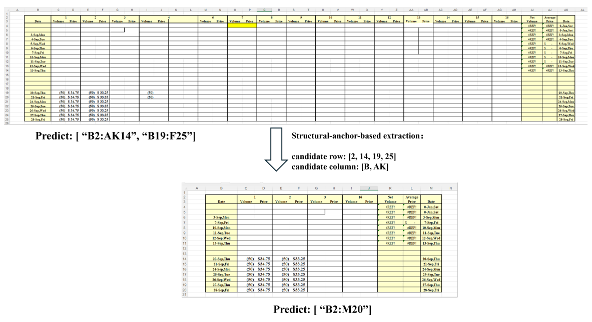

Predict: ["B2:AK14", "B19:F25"]
Structural-anchor-based extraction:
candidate row: [2, 14, 19, 25]
candidate column: [B, AK]
Predict: ["B2:M20"]

Figure 7: The results before and after structural-anchor-based extraction.

The case described in Figure. 7 illustrates the results of GPT4-FT before and after structural-anchor-based extraction. Specifically, before structural-anchor-based extraction, most of the content in the spreadsheet is concentrated in the first two rows and the three columns on the left and right, leaving the middle largely empty. This led GPT4 to incorrectly predict the presence of two tables, "B2:AK14" and "B19:F25." However, after applying structural-anchor-based extraction, the spreadsheet's structure becomes more compact, making it easier for GPT4 to correctly predict the table's range as "B2:M20" after coordinating rearrangement.

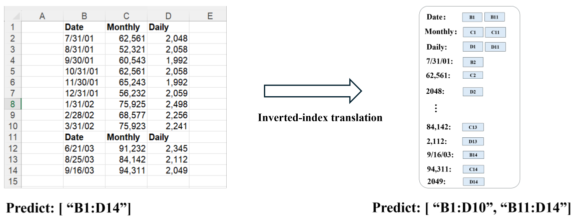
From this case, we can observe that for spread-

Model	Answer	Region
GPT4-compress	0.743	0.974
-w/o Extraction	0.716	0.892
-w/o Translation	0.719	0.925
-w/o Aggregation	0.726	0.928

Table 12: Each module of SHEETENCODER contributes to the positive outcomes on Spreadsheet QA. "Answer" represents the accuracy of answering questions, and "Region" represents the accuracy of predicting relevant regions in CoS.

sheets with sparse structures and many empty cells, structural-anchor-based extraction not only significantly reduces the number of tokens but also effectively enhances GPT4's ability in table detection.

K.2 Comparison of results before and after inverted-index translation



	A	B	C	D	E
1		Date	Monthly	Daily	
2		7/31/01	62,561	2,048	
3		8/31/01	52,321	2,058	
4		9/30/01	60,543	1,992	
5		10/31/01	62,561	2,058	
6		11/30/01	65,243	1,992	
7		12/31/01	56,232	2,059	
8		1/31/02	75,925	2,498	
9		2/28/02	68,577	2,256	
10		3/31/02	75,923	2,241	
11		Date	Monthly	Daily	
12		6/21/03	91,232	2,345	
13		8/25/03	84,142	2,112	
14		9/16/03	94,311	2,049	
15					

Predict: ["B1:D14"]

Predict: ["B1:D10", "B11:D14"]

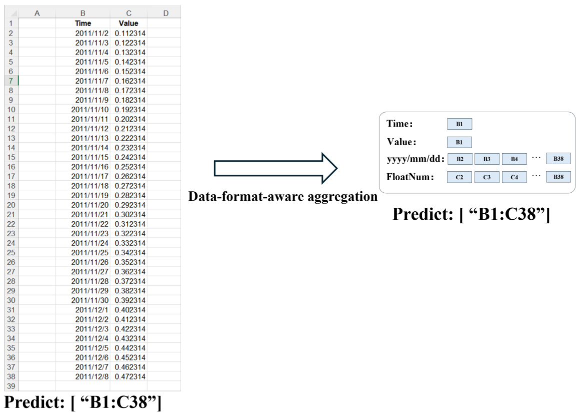
Figure 8: The results before and after inverted-index translation.

The case described in Figure. 8 demonstrates the results of GPT4-FT before and after inverted-index translation. Specifically, before the inverted-index translation, the spreadsheet contained two tables with identical column headers placed closely together, causing GPT4 to mistakenly predict them as one large table, "B1:D14." However, after inverted-index translation, GPT4 was able to aggregate cells with shared values, thereby recognizing semantic relationships between non-adjacent rows and columns. This enabled it to correctly identify the two separate tables in the spreadsheet, "B1:D10" and "B11:D14".

This case indicates that inverted-index translation, by aggregating cells with shared values, not only reduces token redundancy to some extent but also leverages the model's robust understanding of semantic relationships.

K.3 Comparison of results before and after data-format-aware cell aggregation

The case presented in Figure. 9 showcases the results of GPT4-FT before and after data-aware cell aggregation. Specifically, before data-aware cell aggregation, the spreadsheet contained two columns with values of the same data type, occupying a large number of tokens. The first column increased incrementally by date, while the second column increased incrementally by value. After data-aware cell aggregation, the dates in the first column were replaced with the format string "yyyy/mm/dd" and their addresses were aggregated. Similarly, numerical values were handled with a "FloatNum" format. This method allowed the model to predict the table range correctly as "B1:C38," both before and after processing, indicating that this approach significantly reduces the token count while preserving the semantic information of the spreadsheet data.



	A	B	C	D
1		Time	Value	
2		2011/11/2	0.112314	
3		2011/11/3	0.122314	
4		2011/11/4	0.132314	
5		2011/11/5	0.142314	
6		2011/11/6	0.152314	
7		2011/11/7	0.162314	
8		2011/11/8	0.172314	
9		2011/11/9	0.182314	
10		2011/11/10	0.192314	
11		2011/11/11	0.202314	
12		2011/11/12	0.212314	
13		2011/11/13	0.222314	
14		2011/11/14	0.232314	
15		2011/11/15	0.242314	
16		2011/11/16	0.252314	
17		2011/11/17	0.262314	
18		2011/11/18	0.272314	
19		2011/11/19	0.282314	
20		2011/11/20	0.292314	
21		2011/11/21	0.302314	
22		2011/11/22	0.312314	
23		2011/11/23	0.322314	
24		2011/11/24	0.332314	
25		2011/11/25	0.342314	
26		2011/11/26	0.352314	
27		2011/11/27	0.362314	
28		2011/11/28	0.372314	
29		2011/11/29	0.382314	
30		2011/11/30	0.392314	
31		2011/12/1	0.402314	
32		2011/12/2	0.412314	
33		2011/12/3	0.422314	
34		2011/12/4	0.432314	
35		2011/12/5	0.442314	
36		2011/12/6	0.452314	
37		2011/12/7	0.462314	
38		2011/12/8	0.472314	
39				

Predict: ["B1:C38"]

Data-format-aware aggregation
Time: B1
Value: B1
yyyy/mm/dd: B2 B3 B4 ··· B38
FloatNum: C2 C3 C4 ··· B38

Predict: ["B1:C38"]

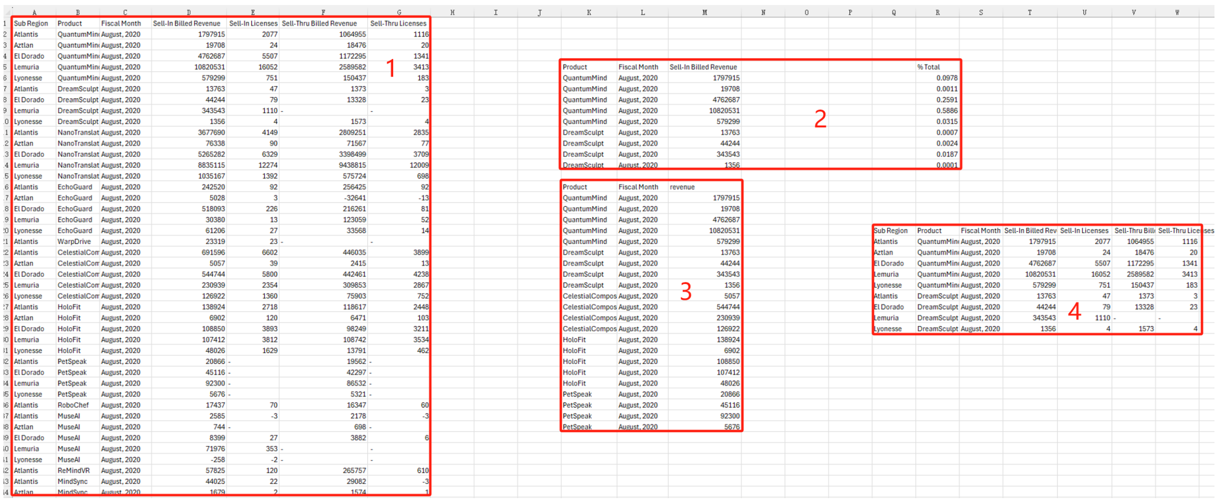
Figure 9: The results before and after data-aware cell aggregation.

K.4 Comparison of SHEETENCODER and TableSense-CNN

As shown in Figure 10, the output of TableSense-CNN is [A1:G44,K5:M14,K16:M38,Q20:W29], while the output of SHEETENCODER is [A1:G44,K5:**R**14,K16:M38,Q20:W29]. SHEETENCODER succeeds in adding the region "R5:R14" to the table2. Though it is spatially distant from the table on the left, SHEETENCODER can extract the connections from cells' semantic and structural relationship, which demonstrates its powerful reasoning ability.

L Prompt Template

In this section, we present the prompt templates for the Spreadsheet Table Detection and Spreadsheet QA tasks.



Sub Region	Product	Fiscal Month	Sell-In Billed Revenue	Sell-In Licenses	Sell-Thru Billed Revenue	Sell-Thru Licenses
Atlantis	QuantumMin	August, 2020	1797915	2077	1064955	1116
Aztlan	QuantumMin	August, 2020	19708	24	18476	20
El Dorado	QuantumMin	August, 2020	4762687	5507	1172295	1341
Lemuria	QuantumMin	August, 2020	10820531	16052	2589582	3413
Lyonesse	QuantumMin	August, 2020	579299	751	150437	183
Atlantis	DreamSculpt	August, 2020	13763	47	1373	3
El Dorado	DreamSculpt	August, 2020	44244	79	13328	23
Lemuria	DreamSculpt	August, 2020	343543	1110	-	-
Lyonesse	DreamSculpt	August, 2020	1356	4	1573	4
Atlantis	NanoTranslat	August, 2020	3677690	4149	2809251	2835
Aztlan	NanoTranslat	August, 2020	76338	90	71567	77
El Dorado	NanoTranslat	August, 2020	5265282	6329	3398499	3709
Lemuria	NanoTranslat	August, 2020	8835115	12274	9438815	12009
Lyonesse	NanoTranslat	August, 2020	1035167	1392	575724	698
Atlantis	EchoGuard	August, 2020	242520	92	256425	92
Aztlan	EchoGuard	August, 2020	5028	3	-32641	-13
El Dorado	EchoGuard	August, 2020	518093	226	216261	81
Lemuria	EchoGuard	August, 2020	30380	13	123059	52
Lyonesse	EchoGuard	August, 2020	61206	27	33568	14
Atlantis	WarpDrive	August, 2020	23319	23	-	-
Atlantis	CelestialCom	August, 2020	691596	6602	446035	3899
Aztlan	CelestialCom	August, 2020	5057	39	2415	13
El Dorado	CelestialCom	August, 2020	544744	5800	442461	4238
Lemuria	CelestialCom	August, 2020	230939	2354	309853	2867
Lyonesse	CelestialCom	August, 2020	126922	1360	75903	752
Atlantis	HoloFit	August, 2020	138924	2718	118617	2448
Aztlan	HoloFit	August, 2020	6902	120	6471	103
El Dorado	HoloFit	August, 2020	108850	3893	98249	3211
Lemuria	HoloFit	August, 2020	107412	3812	108742	3534
Lyonesse	HoloFit	August, 2020	48026	1629	13791	462
Atlantis	PetSpeak	August, 2020	20866	-	19562	-
El Dorado	PetSpeak	August, 2020	45116	-	42297	-
Lemuria	PetSpeak	August, 2020	92300	-	86532	-
Lyonesse	PetSpeak	August, 2020	5676	-	5321	-
Atlantis	RoboChef	August, 2020	17437	70	16347	60
Atlantis	MuseAI	August, 2020	2585	-3	2178	-3
Aztlan	MuseAI	August, 2020	744	-	698	-
El Dorado	MuseAI	August, 2020	8399	27	3882	6
Lemuria	MuseAI	August, 2020	71976	353	-	-
Lyonesse	MuseAI	August, 2020	-258	-2	-	-
Atlantis	ReMindVR	August, 2020	57825	120	265757	610
Atlantis	MindSync	August, 2020	44025	22	29082	-3
Aztlan	MindSync	August, 2020	1679	2	1574	1

Product	Fiscal Month	Sell-In Billed Revenue	% Total
QuantumMind	August, 2020	1797915	0.0978
QuantumMind	August, 2020	19708	0.0011
QuantumMind	August, 2020	4762687	0.2591
QuantumMind	August, 2020	10820531	0.5886
QuantumMind	August, 2020	579299	0.0315
DreamSculpt	August, 2020	13763	0.0007
DreamSculpt	August, 2020	44244	0.0024
DreamSculpt	August, 2020	343543	0.0187
DreamSculpt	August, 2020	1356	0.0001

Product	Fiscal Month	revenue
QuantumMind	August, 2020	1797915
QuantumMind	August, 2020	19708
QuantumMind	August, 2020	4762687
QuantumMind	August, 2020	10820531
QuantumMind	August, 2020	579299
DreamSculpt	August, 2020	13763
DreamSculpt	August, 2020	44244
DreamSculpt	August, 2020	343543
DreamSculpt	August, 2020	1356
CelestialCompos	August, 2020	5057
CelestialCompos	August, 2020	544744
CelestialCompos	August, 2020	230939
CelestialCompos	August, 2020	126922
HoloFit	August, 2020	138924
HoloFit	August, 2020	6902
HoloFit	August, 2020	108850
HoloFit	August, 2020	107412
HoloFit	August, 2020	48026
PetSpeak	August, 2020	20866
PetSpeak	August, 2020	45116
PetSpeak	August, 2020	92300
PetSpeak	August, 2020	5676

Sub Region	Product	Fiscal Month	Sell-In Billed Rev	Sell-In Licenses	Sell-Thru Bill	Sell-Thru Licenses
Atlantis	QuantumMin	August, 2020	1797915	2077	1064955	1116
Aztlan	QuantumMin	August, 2020	19708	24	18476	20
El Dorado	QuantumMin	August, 2020	4762687	5507	1172295	1341
Lemuria	QuantumMin	August, 2020	10820531	16052	2589582	3413
Lyonesse	QuantumMin	August, 2020	579299	751	150437	183
Atlantis	DreamSculpt	August, 2020	13763	47	1373	3
El Dorado	DreamSculpt	August, 2020	44244	79	13328	23
Lemuria	DreamSculpt	August, 2020	343543	1110	-	-
Lyonesse	DreamSculpt	August, 2020	1356	4	1573	4

Figure 10: A challenging case. Traditional spreadsheet understanding methods usually miss the region "R5:R14", but this column has a semantic relationship with the left cells, representing the percentage of those values in left cells.

L.1 Vanilla Prompt Template for Spreadsheet Table Detection

A Vanilla Prompt Template for Spreadsheet Table Detection:

```
INSTRUCTION:
Given an input that is a string denoting data of cells in a spreadsheet. The input spreadsheet includes many pairs, and each pair consists of a cell address and the text in that cell with a ',' in between, like 'A1,Year'. Cells are separated by '|' like 'A1,Year|A2,Profit'. The text can be empty so the cell data is like 'A1, |A2,Profit'. The cells are organized in row-major order. Now you should tell me the range of the table in a format like A2:D5, and the range of the table should only CONTAIN HEADER REGION and the data region, DON'T include the title or comments. Note that there can be more than one table in the string, so you should return all the RANGE, LIKE ['range': 'A1:F9', 'range': 'A12:F18']. DON'T ADD OTHER WORDS OR EXPLANATION.
INPUT:
[Encoded Spreadsheet]
```

L.2 Prompt Template for Spreadsheet Table Detection

SheetEncoder Prompt Template for Spreadsheet Table Detection:

```
INSTRUCTION:
Given an input that is a string denoting data of cells in an Excel spreadsheet. The input spreadsheet contains many tuples, describing the cells with content in the spreadsheet. Each tuple consists of two elements separated by a '|': the cell content and the cell address/region, like (Year|A1), ( |A1) or (IntNum|A1:B3). The content in some cells such as '#,##0'/'d-mmm-yy'/'H:mm:ss',etc., represents the CELL DATA FORMATS of Excel. The content in some cells such as 'IntNum'/'DateData'/'EmailData',etc., represents a category of data with the same format and similar semantics. For example, 'IntNum' represents integer type data, and 'ScientificNum' represents scientific notation type data. 'A1:B3' represents a region in a spreadsheet, from the first row to the third row and from column A to column B. Some cells with empty content in the spreadsheet are not entered. Now you should tell me the range of the table in a format like A2:D5, and the range of the table should only CONTAIN HEADER REGION and the data region. DON'T include the title or comments. Note that there can be more than one table in a string, so you should return all the RANGE.
```

DON'T ADD OTHER WORDS OR EXPLANATION.
INPUT:
[Encoded Spreadsheet]

L.3 Prompt Template for Spreadsheet QA

As detailed in Section 4.2, the CoS method includes two stages, and the prompts for each stage are as follows:

Spreadsheet QA Prompt Template:

Stage 1:
INSTRUCTION:
Given an input that is a string denoting data of cells in a table. The input table contains many tuples, describing the cells with content in the spreadsheet. Each tuple consists of two elements separated by a '|': the cell content and the cell address/region, like (Year|A1), (|A1) or (IntNum|A1:B3). The content in some cells such as '#,##0'/'d-mmm-yy'/'H:mm:ss',etc., represents the CELL DATA FORMATS of Excel. The content in some cells such as 'IntNum'/'DateData'/'EmailData',etc., represents a category of data with the same format and similar semantics. For example, 'IntNum' represents integer type data, and 'ScientificNum' represents scientific notation type data. 'A1:B3' represents a region in a spreadsheet, from the first row to the third row and from column A to column B. Some cells with empty content in the spreadsheet are not entered. How many tables are there in the spreadsheet? Below is a question about one certain table in this spreadsheet. I need you to determine in which table the answer to the following question can be found, and return the RANGE of the ONE table you choose, LIKE ['range': 'A1:F9']. DON'T ADD OTHER WORDS OR EXPLANATION.
INPUT:
[Encoded Spreadsheet with compression]

Stage 2:
INSTRUCTION:
Given an input that is a string denoting data of cells in a table and a question about this table. The answer to the question can be found in the table. The input table includes many pairs, and each pair consists of a cell address and the text in that cell with a ',' in between, like 'A1,Year'. Cells are separated by '|' like 'A1,Year|A2,Profit'. The text can be empty so the cell data is like 'A1, |A2,Profit'. The cells are organized in row-major order. The answer to the input question is contained in the input table and can be represented by cell address. I need you to find the cell address of the answer in the given table based on the given question description, and return the cell ADDRESS of the answer like '[B3]' or '[SUM(A2:A10)]'. DON'T ADD ANY OTHER WORDS."
INPUT:
[Encoded Spreadsheet without compression]

M Algorithm Steps

M.1 Identical Cell Aggregation

The corresponding algorithm steps is shown in Algorithm 1.

Algorithm 1: Identical Cell Aggregation

```
Input  : Matrix $nfs$ composed of all cell values in the spreadsheet.
1  Initialize $m$ and $n$ as the number of matrix $input$ rows and columns.
2  Initialize the $m \times n$ matrix $visited$ with all values set to $False$.
3  Initialize $areas$ as an empty list.
4  Initialize the $FormatDict$ dictionary, the key-value pairs are data values and predefined types respectively.
5  Function dfs($r$, $c$, $val\_type$):
6      if $visited[r][c]) \vee val\_type! = FormatDict[nfs[r,c]]$ then
7          return $[r, c, r-1, c-1]$;
8      $visited[r][c] \leftarrow$ True;
9      $bounds \leftarrow [r, c, r, c]$;
10     foreach $(tr, tc)$ around $(r, c)$ do
11         if $\neg visited[tr][tc]) \wedge val\_type == FormatDict[nfs[tr, tc]]$ then
12             $new\_bounds \leftarrow$ dfs$(tr, tc, val\_type)$;
13             update bounds to include new_bounds;
14     return bounds;
15 for $r = 0$ to $m-1$ do
16     for $c = 0$ to $n-1$ do
17         if $\neg visited[r][c]$ then
18             $val\_type \leftarrow$ FormatDict$[nfs[r,c]]$;
19             $bounds \leftarrow$ dfs$(r, c, val\_type)$;
20             $areas \leftarrow areas + ((bounds[0], bounds[1]),$
21             $(bounds[2], bounds[3]),$
22             $val\_type)$;
Output : Aggregation matrix $areas$, each cell which is filled with the corresponding datatype after applying custom rules.
```

M.2 Table Split QA Algorithm

The corresponding algorithm steps are shown in Algorithm 2.

Algorithm 2: Question Answering Process for Large Tables

```
Input  : $question$ composed of strings and two-dimensional matrix $region$
1  Initialize $header$ and $answers$ to empty lists
2  if $calculateTokens(region) \leq 4096$ then
3      return answer_question(question, region);
4  else
5      $header \leftarrow predict\_header(region)$;
6      $body \leftarrow region[length(header) + 1 : end]$;
7      for $i = 0$ to $length(body)$ do
8          $new\_table \leftarrow header + body[i : i + 3]$;
9          $answer \leftarrow answer\_question(question, table)$;
10         $answers.append(answer)$;
Output : final result $answers$
```