

Usage of Large Language Models in Structured Data Analysis

Konstantin Burlachko¹ and Boris Dobrov²

¹ Lomonosov Moscow State University, Moscow, Russia
`cburlachko@gmail.com`

² Lomonosov Moscow State University, Moscow, Russia
`dobrov_bv@rcc.msu.ru`

Abstract. The rapid advancement of Large Language Models (LLMs) has created new opportunities for automating tasks involving tabular data, including spreadsheet operations and data analysis. However, applying LLMs to this domain faces significant challenges, including context limitations, insufficient proficiency in numerical calculations, and the lack of standardized evaluation methods. This paper explores the current state of LLM applications for tabular data, reviewing both agent-based and code-generation approaches. We evaluate performance using standard metrics to assess the robustness and correctness of the generated solutions. Experimental results show that modern LLMs achieve a high degree of accuracy in generating executable code for tabular data tasks. Our findings highlight the potential of LLMs to transform tabular data management by overcoming existing limitations with innovative workflows and architectural solutions.

Keywords: Large language Models · Spreadsheet Automation · Data Analysis · Code generation

1 Introduction

Tabular data is a core of modern information management. It spreads throughout many fields such as finance, scientific research, commerce, marketing. Common tasks comprising tabular data processing and analysis typically performed in spreadsheet applications such as Microsoft Excel and Google Sheets are essential for work effectiveness [9]. Yet, these tasks are trivial and time-consuming and typically demand specialized skills. This can be a struggle for some end-users. Automating such routine work can reduce manual effort significantly and enhance efficiency.

The recent development of Large Language Models (LLMs) brings us closer to the goal of intuitively controlling advanced software using natural language. Applying LLMs to the field of tabular data, particularly spreadsheet operations and data analysis, has been a subject of significant interest. However, using LLMs directly to read and process large-scale data encounters a number of major challenges:

First, context limitations: due to their context length limits, LLMs are not able to read and process large-scale data directly [8]. This also poses a potential risk of data leakage when LLMs directly access private data sources.

Second, insufficient proficiency in numerical calculation and table operation: LLMs are not inherently capable of carrying out numerical calculations and complicated table operations [7,10]

Third, lack of standardized benchmarks and frameworks: Progress in this field has been slowed by the lack of frameworks that manage model-application interaction, furthermore the lack of extensive benchmarks for performance evaluation on actual and challenging tasks. Current benchmarks tend to concentrate on narrow task subsets or over-simplified cases.

Fourth, tables and text differ greatly from one another. The key differences between them are described by Li et al. [5]. Texts are (1) a single direction; (2) usually read from left to right; and (3) the meaning of a sentence is usually altered when two tokens swap places. Tables, on the other hand, are (1) two-dimensional and must be read both horizontally and vertically; (2) schemas or header names are crucial to their comprehension; and (3) some of them are not impacted by permutations of rows and columns.

This study systematically analyzes LLMs for managing tabular data by evaluating both agent-based and code-generation approaches. Section 2 reviews related works, exploring various LLM approaches such as code generation and agent-based control for tabular data analysis. Section 3 details the datasets used, including the SheetCopilot dataset and the authors’ modifications, which focus on business, commerce, and finance domains. Section 4 presents the proposed method, which leverages LLMs to generate executable VBA code from natural language specifications, emphasizing prompt design for efficiency and accuracy. Section 5 discusses experimental results, evaluating LLM performance using Exec@1 and Pass@1 metrics across different task categories and comparing models like QWEN2.5-72b, ChatGPT4, and SheetCopilot. Finally, Section 6 concludes the paper by summarizing key findings, highlighting barriers, reviewing solutions.

2 Related Works

The rapid advancement in Large Language Models (LLMs) has significantly expanded their prospective applications in various fields, including the complex area of data analysis and manipulation, specifically in relation to tabular data such as spreadsheets. Research efforts have thus explored various approaches to using LLMs for analyzing tabular data with a focus on code generation agent-based control, and richer data comprehension.

One of the key areas of research is to utilize LLMs for general query and analysis of data from structured data sources like databases and tables. This includes their application in Text2SQL tasks, where the goal is to translate natural language queries into executable SQL code.

Approaches like StructGPT [12] have been introduced as general frameworks that leverage LLMs for reasoning over structured data. Some engage in automatic data exploration, while others, like Data-Copilot [6], employ LLMs in a code-centric manner to generate code as an intermediate to visualize and process massive data based on human input. Data-Copilot addresses key challenges by incorporating a step of data exploration to synthesize tested code modules (interfaces) that are callable for real-time requests in order to reduce errors compared to code generation from scratch. This method includes parsing data sources to make the LLM learn the structure and contents of the data.

Another domain is the development of LLM-based agents to control software applications, including spreadsheet software. Tool-augmented research integrates third-party tools and LLMs to enhance their capability, ranging from web browsers to tool libraries. For spreadsheets, agents package functionality into sequences of actions or generate code to interact with the application. SheetCopilot [3], for instance, is being proposed as an agent that takes natural language tasks and operates on spreadsheet software by decomposing high-level tasks into step-by-step solutions in terms of atomic actions as abstractions. SheetAgent [11] is another standalone agent with LLMs to reason and manipulate spreadsheets to solve complex and realistic tasks comprising reasoning problems. These agents use internal methods like state machines for task planning or recursive task reasoning and reflection mechanisms.

Another concept is control through software. Those works also aim at LLMs for spreadsheet-specific operations and code generation specifically. This includes code generation, e.g., Excel OfficeScripts or Visual Basic for Applications (VBA), from natural language instructions directly. Recent work has investigated deep learning methods to automate certain tasks such as formatting and formula prediction; these methods might not cover a general set of operations. The advent of large-capacity LLMs has witnessed their application in generating intricate Excel formulas from natural language descriptions and code generation for specialized APIs like Microsoft Excel’s OfficeScripts. Code generation for these types of niche APIs could be challenging because LLMs would be less likely to have encountered them in their pre-training data to the same extent that they do for general-purpose languages. One other solution is the generation of VBA code, yet this is limited by the problems of the frequency of VBA code in LLM training corpora and the variability of software support.

Despite ongoing research in the field of tabular data manipulation using LLMs, a definitive solution to this task has not yet been found, and there is no consensus among scientists about the best method. Yet, direct Text2Code approaches potentially provides solution for a wide range of tasks, not limited to a specific, predefined API.

3 Datasets

Evaluating natural language interfaces to control software, and in particular spreadsheet manipulation, needs a standard benchmark. Previous work has prima-

rily addressed a limited range of tasks, a prominent example being formula generation.

Because of this, the SheetCopilot [3] project presented a dataset and evaluation framework aimed at assessing LLMs sheet control capabilities. The dataset was compiled by scraping tasks and worksheets from the internet, specifically from approximately 16,000 question and answer pairs concerning spreadsheets at www.superuser.com. The initial set was reduced to approximately 13,000 task-relevant pairs and sorted into six types. Example "seed tasks" (67 were selected) were identified. For the provision of realistic context, 28 real spreadsheets were compiled as a "workbench". The GPT-4 model was employed to translate the seed tasks according to the evaluation sheets to produce new instructions that were then condensed to be suitable for non-expert user language. The resulting core dataset includes 221 varied spreadsheet control tasks.

To address more specific, yet widely used tasks and transformation types, we propose a modification of the SCB dataset. First, seven new files from the financial domain have been introduced, for which a total of 50 questions have been formulated. Second, we suggest removing tasks that are least likely to be encountered by a potential user in practical applications (e.g. physics domain). New set of tasks focuses on business analysis, commerce, and finance. The selection reflects domain-specific concepts, emphasizing the use of standard financial terms, metrics, and indicators. Furthermore, several tasks are designed to simulate authentic end-to-end data analysis workflows. Initial and modified datasets are published on [2].

A comparison by category of the initial "SheetCopilotBench" and resulting datasets listed in Table 1.

Table 1. Datasets task categories distribution

Category	SheetCopilotBench	Proposed dataset
Charts	53	59
Entry and manipulation	165	169
Pivot Table	39	34
Management	24	27
Formatting	65	60
Formula	97	95

4 Proposed Method

Our system leverages large language models (LLMs) to execute tabular data management tasks automatically through generating executable VBA code from natural language specifications. The system prompt is designed to guide the LLM in generating efficient and accurate VBA scripts with few errors. The prompt design integrates key aspects of SheetCopilot and InstructExcel, modified for direct VBA generation.

First, task specification: The prompt clearly states the task (e.g., data filtering, formula application, or report generation) with formatted input-output examples.

Second, contextual guidance: The prompt supplies schema information (column names, data types) and constraints (e.g., support for particular Excel versions) to inform syntactically correct VBA.

Third, prevention of errors: Typical errors (e.g., erroneous range references or loop constructs) are prevented ahead of time through table constraints.

In contrast to SheetCopilot, which emphasizes advanced spreadsheet functionality, and InstructExcel, which employs a special-purpose programming language, our approach generates directly executable VBA code. This direct VBA generation enables more Excel automation but remains interpretable. The generated code is verified prior to deployment by executing it in a controlled sandboxed environment.

5 Experimental Results

In order to assess the capability of large language models (LLMs) to generate VBA code for manipulating spreadsheets, we propose an evaluation approach that assesses the LLM’s ability to convert natural language into correct and robust executable code. The evaluation considers the correctness and robustness of the generated VBA code, and we follow an execution-based evaluation from the perspective of executing the generated code on the target spreadsheets.

Exec@1 : This metric calculates the proportion of generated plans or solutions that execute without throwing exceptions or errors. The higher Exec@1, the more robust the agent’s generated plans are and the less likely they are to fail to execute.

Moreover, once we execute the VBA code on each test case, the spreadsheet must have the same final state as the ground truth output for that specific test case. This ensures that the code ran, and it completed the task across different data instances to assess simple robustness.

Pass@1 : This quantifies the functional correctness of a generated solution. A plan or solution is deemed correct if the final state of the spreadsheet under test satisfies task requirements in full. In other words, a high Pass@1 score will mean the agent can successfully complete the user’s requested task accurately.

Exec@1 and Pass@1 are usually expressed as a percentage, with higher percentages indicating better performance. Pass@1 and Exec@1 are separate measures: Exec@1 will focus simply on whether something that could be run and executed without crashing, while Pass@1 will be focused on whether that task was completed successfully and correctly according to the original requirements.

For the purpose of comparison, the following approaches using QWEN2.5-72b [11], ChatGPT4 [1] and LLaMA3-70b [4] were selected: - ChatGPT4: This

approach uses custom prompt including sheet state description. The temperature and top p of web version is not publicly available.

- QWEN2.5-72b: This approach uses custom prompt including sheet state description. The temperature was adjusted to 0.1, and top p was set to 0.9 to encourage correctness and consistency of generated code.

- SheetCopilot framework, based on LLaMA3-70b: This approach utilizes basic framework, including sheet state description in standardized initial format usage. The temperature also was adjusted to 0.1, and top p was set to 0.9.

This list of examined approaches cover both reviewed methods: LLM as a planner and LLM as a code generator, and variety of modern base LLM models.

In order to demonstrate common differences between those approaches, we provide brief pipeline with some examples for relatively easy (all three approaches solved a problem correctly) and difficult tasks (only one approach solved a problem correctly). Fig. 1 shows common prompt technic for code-generation based methods (ChatGPT4 and QWEN2.5-72b). Prompt was carefully designed to provide LLM with all crucial information about initial task, restrictions on programming language, file state, and exogenous variables restriction.

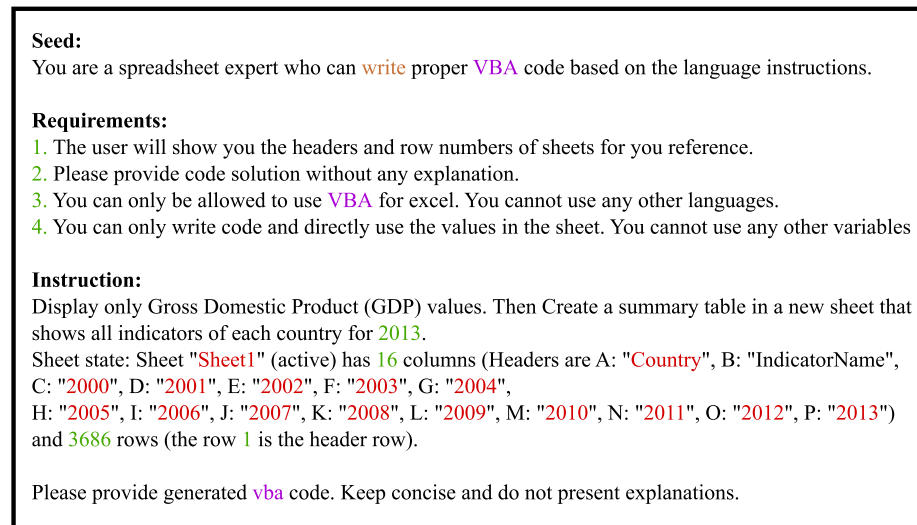


Fig. 1. Example of prompt with initial instruction for code generating approach. Prompt consist of initial seed message and task specific instruction with information about source Excel file (sheet state)

As a result of code-generating approach we expect a valid VBA code, which can be later executed in Excel directly to solve initial problem. The examples of generated VBA code for a given example task from Fig. 1 are presented in Fig. 2 for ChatGPT-4 case QWEN2.5-72b case respectively.

In case of ChatGPT4, code firstly filters "Sheet1" to show only rows where "Indicator Name" is "Gross Domestic Product (GDP)". Creates a new sheet named "Summary2013", copying all indicators (not just GDP) for all countries where 2013 data is available, into that new sheet, hiding non-GDP rows by collapsing them. QWEN2.5-72b code is looping through all rows in "Sheet1" without filtering them, which leaves first part of task unsatisfied. While looping rows, code copies "Country Name", "Indicator Name" and "2013" column value. These leads to collecting only a small part of necessary data.

<pre> Sub DisplayGDPAndCreateSummary() Dim ws As Worksheet Dim summaryWs As Worksheet Dim lastRow As Long Dim i As Long Dim summaryRow As Long Set ws = ThisWorkbook.Sheets("Sheet1") Set summaryWs = ThisWorkbook.Sheets.Add(After:=ws) summaryWs.Name = "Summary" lastRow = ws.Cells(ws.Rows.Count, 1).End(xlUp).Row summaryRow = 1 summaryWs.Cells(1, 1).Value = "Country" summaryWs.Cells(1, 2).Value = "IndicatorName" summaryWs.Cells(1, 3).Value = "2013" For i = 2 To lastRow If ws.Cells(i, 2).Value = "Gross Domestic Product (GDP)" Then ws.Rows(i).Hidden = False Else ws.Rows(i).Hidden = True End If If ws.Cells(i, 2).Value = "Gross Domestic Product (GDP)" Then summaryRow = summaryRow + 1 summaryWs.Cells(summaryRow, 1).Value = ws.Cells(i, 1).Value summaryWs.Cells(summaryRow, 2).Value = ws.Cells(i, 2).Value summaryWs.Cells(summaryRow, 3).Value = ws.Cells(i, 16).Value End If Next i End Sub </pre>	<pre> Sub CreateGDP2013Summary() Dim ws As Worksheet Set ws = ThisWorkbook.Sheets.Add ws.Name = "GDP 2013 Summary" Dim lastRow As Long lastRow = ThisWorkbook.Sheets("Sheet1").Cells(Rows.Count, 1).End(xlUp).Row ws.Cells(1, 1).Value = "Country" ws.Cells(1, 2).Value = "Indicator Name" ws.Cells(1, 3).Value = "2013 Value" Dim row As Long row = 2 Dim i As Long For i = 2 To lastRow If Cells(i, 2).Value = "Gross Domestic Product (GDP)" Then ws.Cells(row, 1).Value = Cells(i, 1).Value ws.Cells(row, 2).Value = Cells(i, 2).Value ws.Cells(row, 3).Value = Cells(i, 16).Value row = row + 1 End If Next i End Sub </pre>
---	---

Fig. 2. Examples of generated code for a given task. QWEN2.5-72b code (left) with correct initial data filtering and collecting all necessary columns and ChatGPT4 code (right) with noticeable mistakes in filtering and collecting data in terms of given task.

Despite the fact that both language models generated valid and executable VBA code, i.e. passed the run test (exec@1), both models failed to fulfill all conditions of the original task: "Display only Gross Domestic Product (GDP) values" and "Create a summary table in a new sheet that shows all indicators of each country for 2013". Thus, ChatGPT4 did not allocate only the 2013 column to a separate sheet, and QWEN2.5-72b did not add all rows from the "IndicatorName" column. Brief results of executing generated VBA code are presented in Table 2 and Table 3.

We conduct experiments on six common tabular tasks: charts, entry and manipulation, pivot table, management, formatting, and formulas insert. To provide a clearer understanding of the impact of different categories of tasks, we conducted experiments as shown in Table 4

The results show that the overall accuracy is higher in case of using QWEN2.5-72b with VBA code generation approach, also this approach shows significantly

Table 2. Part of QWEN2.5-72b solution Excel tables result with skipped Indicator-Name columns entries.

	Country	IndicatorName	2013
1	Afghanistan	Gross ... (GDP)	13 336 226 877
2	Albania	Gross ... (GDP)	10 996 683 701
3	Algeria	Gross ... (GDP)	127 689 050 690
4	Andorra	Gross ... (GDP)	2 644 100 013
5	Angola	Gross ... (GDP)	66 346 877 923
...	...	Gross ... (GDP)	...

Table 3. Part of ChatGPT4 solution Excel tables result with excessive year columns. Filtered (hidden) rows marked with blue color.

	Country	IndicatorName	2000	...	2013
...	...	...	...	...	...
9	Afghanistan	Gross ... (GDP)	3 495 246 010	...	13 336 226 877
10	Afghanistan	Agriculture...	1 937 654 304	...	3 083 723 843
11	Afghanistan	Mining...	607 223 870	...	1 525 274 415
...	...	...	...	...	...

higher ratio Pass@1 to Exec@1, which corresponds to producing more robust pipeline of task solving. We hypothesize that QWEN2.5-72b strong performance is due to its extensive training on code corpora, which may have included a significant amount of VBA syntax, giving it an advantage in generating logically, syntactically correct code for this specific domain compared to the more general-purpose ChatGPT-4

From the perspective of various typical tabular data tasks, creating and managing pivot tables remains most difficult problem for approaches based on code generation. Creating a pivot table is not a single action but a multi-step process (defining data range, rows, columns, values, filters). We believe, LLMs struggle with this compositional reasoning. Furthermore, the VBA API for pivot tables (PivotCaches, PivotFields) is complex and less common in training data than simple Range operations. On the other hand, Tasks like sorting, filtering, or moving columns are often atomic, well-defined, and have straightforward VBA equivalents (Sort, AutoFilter, Copy/Paste) and over-presence in online forums and tutorials means they are heavily represented in LLM pre-training data. Chart creation in VBA involves numerous aesthetic properties (Chart-Type, ChartStyle, HasTitle, etc.). Code-generation models often produce code that is functionally correct (creates a chart) but fails on precise specifications (e.g., wrong chart type, missing titles, incorrect data range selection). This leads to a Pass@1 failure as the output doesn't fully meet the requirement.

Table 4. Models performance over different categories of tasks

Model	Category	Exec@1	Pass@1
QWEN2.5-72b	Charts	81,4%	76,3%
	Entry and manipulation	87,6%	83,4%
	Pivot Table	76,5%	76,5%
	Management	96,3%	92,6%
	Formatting	86,7%	81,7%
	Formula	88,4%	85,3%
ChatGPT4	Charts	69,5%	55,9%
	Entry and manipulation	79,3%	65,1%
	Pivot Table	64,7%	55,9%
	Management	96,3%	85,2%
	Formatting	81,7%	70%
	Formula	84,2%	71,6%
SheetCopilot	Charts	66,1%	42,4%
	Entry and manipulation	69,2%	47,9%
	Pivot Table	73,5%	50,0%
	Management	81,5%	74,1%
	Formatting	66,7%	50%
	Formula	72,6%	50,5%

6 Conclusion

Paper shows, that usage of Large Language Models for tabular data management have the potential to automate repetitive and complex tasks, increasing efficiency and decreasing manual labor.

Paper revise potential solutions including agents to direct invocations or generate code directly; and conducted experiments suggesting LLMs such as QWEN2.5-72b can generate correct, executable VBA code with Pass@1 rates of 85.1%, highlighting current difficulties of processing different types of tasks associated with tabular data and spreadsheet manipulation.

Moving beyond performance metrics, our work highlights that completing tasks depends on the semantic complexity and API specificity of the tasks. Simple, atomic tasks (e.g., data sorting, formatting, formula insertion) are handled with high reliability, as they benefit from a direct instruction-to-code mapping and are well-represented in training data. Complex, multi-step tasks (e.g., pivot table creation) remain a significant challenge. Failures here are primarily due to logical errors in task decomposition.

However, more work needs to be done related to benchmarks and numeracy. Current investigations in this space shows the potential of LLMs to democratize access to advanced manipulation techniques to non-technical users. Future work should continue to improve the capabilities of models, and expand the tasks they take on to real-world environments.

Acknowledgments

The study was conducted under the state assignment of Lomonosov Moscow State University.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Burlachko, K., Dobrov, B.: Dataset of excel tasks for llms. Zenodo (2025). <https://doi.org/10.5281/zenodo.17209399>, data set
3. Chen, Y., Yuan, Y., Zhang, Z., Zheng, Y., Liu, J., Ni, F., Hao, J.: Sheetagnt: A generalist agent for spreadsheet reasoning and manipulation via large language models. In: ICML 2024 Workshop on LLMs and Cognition (2024)
4. Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al.: The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024)
5. Jiang, J., Zhou, K., Dong, Z., Ye, K., Zhao, W.X., Wen, J.R.: Structgpt: A general framework for large language model to reason over structured data. arXiv preprint arXiv:2305.09645 (2023)
6. Li, H., Su, J., Chen, Y., Li, Q., Zhang, Z.X.: Sheetcopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems* **36**, 4952–4984 (2023)
7. Li, P., He, Y., Yashar, D., Cui, W., Ge, S., Zhang, H., Rifinski Fainman, D., Zhang, D., Chaudhuri, S.: Table-gpt: Table fine-tuned gpt for diverse table tasks. *Proceedings of the ACM on Management of Data* **2**(3), 1–28 (2024)
8. Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use long contexts. arXiv preprint arXiv:2307.03172 (2023)
9. Rahman, S., Mack, K., Bendre, M., Zhang, R., Karahalios, K., Parameswaran, A.: Benchmarking spreadsheet systems. In: *Proceedings of the 2020 acm sigmod international conference on management of data*. pp. 1589–1599 (2020)
10. Sui, Y., Zhou, M., Zhou, M., Han, S., Zhang, D.: Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In: *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. pp. 645–654 (2024)
11. Team, Q.: Qwen2 technical report. arXiv preprint arXiv:2407.10671 (2024)
12. Zhang, W., Shen, Y., Lu, W., Zhuang, Y.: Data-copilot: Bridging billions of data and humans with autonomous workflow. arXiv preprint arXiv:2306.07209 (2023)