A *workbook time machine* for spreadsheet creation benchmarks

Anonymous ACL submission

Abstract

We introduce the *workbook time machine* for automatically creating benchmarks that evaluate the ability of language models to create (sequences of) calculated objects (formulas, charts, pivot tables, and conditional formatting) in spreadsheets. We generate and select 262 problems on public workbooks that require a different number of intermediate steps (like formula → chart) and generate different instructions of increasing abstractness. We evaluate existing spreadsheet manipulation agents and baselines on these two different dimensions, as well as their ability to generate different types of objects. Our evaluation shows that straight code generation outperforms agents on simple problems and problems with detailed instructions, and that the API used to control spreadsheets pose a significant limitation, trading off easy-of-use (Python) for completeness (VBA).

1 Introduction

Creating and manipulating complex spreadsheets is challenging for users, but it is even more challenging for large language models: SpreadsheetBench [5] and SheetCopilotBench [4] report 20% and 55% success rates on their respectively proposed benchmarks using the best performing strategy. Human subjects scored 71% on SpreadsheetBench.

Unfortunately, these benchmarks do not paint a realistic picture of real-world spreadsheet creation tasks, either considering the creation of derived objects like charts and pivot tables but only in simple workbooks (SheetCopilotBench) or considering more realistic workbooks from users but only considering data entry and manipulation (SpreadsheetAgent).

In this paper, we introduce the *workbook time machine* for automatically generating benchmarks for creating derived objects in spreadsheets. Given a spreadsheet, we go back in time and remove derived objects that users have created (like charts and formulas) and associated metadata (like a description of the formula). We then go forward in time and generate instructions in natural language of what task to achieve.

Example 1 *Figure 1 shows part of a spreadsheet with three derived artifact that make up two sequences: formula → conditional formatting and formula → chart. In the backwards pass, we remove the derived artifacts to obtain the initial spreadsheet. In the forward pass, we generate an instruction for one such sequence of two steps ("plot BMI versus age").*

This flexible process yields benchmarks that (1) we can evaluate based on the removed final artifact, (2) vary in complexity by considering fewer or more steps, (3) vary in complexity by generating descriptions with less or more details.

We evaluate existing spreadsheet agents on our benchmark and show that (1) simple baselines work well with elaborate instructions and agents work better with high-level instructions, (2) the choice of API trades off ease-of-use (Python) for completeness (VBA).

We make the following contributions:

- We introduce the *workbook time machine* for generating sequences of spreadsheet actions that achieve real-world tasks, as well as instructions that describe these tasks.
- We compare existing inference-based spreadsheet agents on the generated benchmarks and analyze their performance.
- We release the generated benchmark and evaluation to encourage further research on spreadsheet manipulation.

2 Related Works

Table 1 summarizes properties of existing benchmarks and our WTM benchmark.

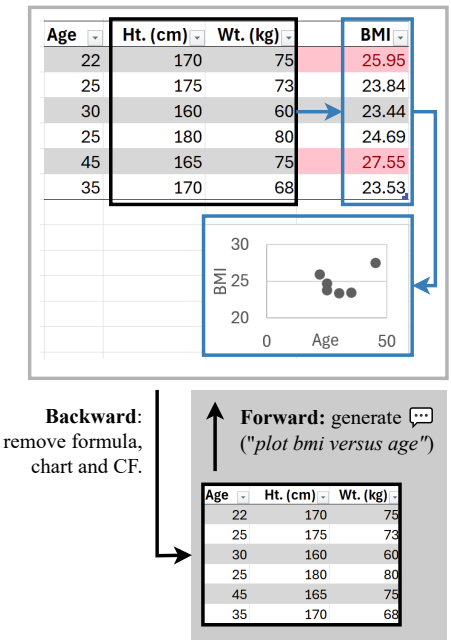


Figure 1: Example of creating a benchmark (in gray) with the *workbook time machine*, which removes objects from a workbook and then generates an instruction that describes the removed objects.

Table 1: Comparison of properties of the input (**I**), workbook (**W**) and output (**O**) of SheetCopilotBench (**SCB**) [4], InstructExcel (**IE**) [6], SheetRM (**SRM**) [2], SpreadsheetBench (**SB**) [5] and our **WTM** benchmark. * Paper mentions yes, not found in practice in the dataset.

		SCB	IE	SRM	SB	WTM
I	#	221	4850	201	912	262
	levels	1	1	1	1	5
W	real?	✓	✓	✘	✓	✓
	+ sheets?	✓	✓	✘	✓	✓
	+ tables?	✘	✓	✘	✓	✓
	+ info?	✘	✓	✘	✘	✓
O	formulas	✓	✓	✓	✓	✓
	charts	✓	✓	✓	✘*	✓
	PT	✓	✓	✘*	✘*	✓
	CF	✓	✓	✓	✓	✓
	cell manip.	✓	✓	✓	✓	✘
	comb.	✓	✘	✓	✓	✓

3 Workbook Time Machine

The workbook time machine performs two steps: a backward step in which objects are removed, and a forward step in which instructions are generated.

3.1 Backward

In the backward step, we iteratively remove derived artifacts that are not required by any other derived artifacts. The artifacts considered are formulas, conditional formatting, charts, and pivot tables.

We require the following special considerations:

- Adjacent formulas with the same formula template after anonymizing cell references are grouped and considered as a single formula.
- When removing a column in a defined table (created using Insert → Table) those cells are deleted and all cells to the right of the table are shifted to their original position.

Cleaning formula descriptors A common occurrence are cells that describe a surrounding formula, for example, a "total" cell next to `=SUM()`. These descriptors leak information about how to solve the task. We use two subsequent LLM cleaning steps, where we ask the model to identify cells that do not refer to any relevant information.

Cleaning workbooks To keep the spreadsheets reasonably sized and focus on the ability to solve the task, workbooks and defined tables that are not used in any derived artifact are removed.

3.2 Forward

In the forward step, we generate natural language instructions that describe the objects that should be added to the workbook. For each intermediate object from the original workbook, we generate five instructions with decreasing detail using an LLM (OpenAI `gpt-4o`). The context to generate these instructions contains a Markdown representation of the before and after states, as well as the address of the relevant range. An example of five instructions for the same problem is shown in Table 2.

3.3 Dataset statistics

We use the workbook time machine on two public corpora (FUSE [1] and Enron [3]) and one private set of publicly available workbooks collected from the web. Figure 2 shows information about the workbooks, artifacts, and user trajectories that our dataset consists of.

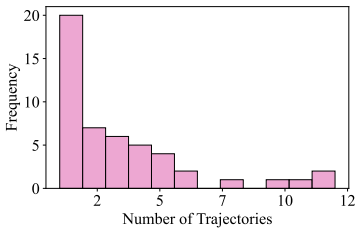
4 Experiment Setup

We evaluate baselines and existing approaches on our dataset.

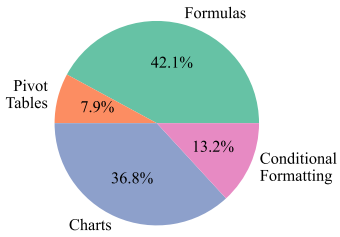
- **SheetCopilot** [4] is a specialized agent that uses `xlwings` to interact with the spreadsheet.
- **SheetAgent** [2] is a specialized agent that uses `openpyxl` to interact with the spreadsheet.

Table 2: Different levels of instructions with increasing levels of abstractness.

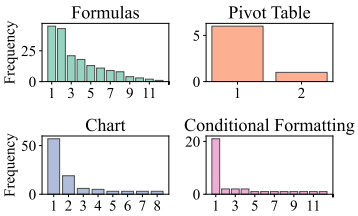
1	In Sheet1, calculate the BMI for each individual by using the formula BMI = 10000 * Wt. (kg) / (Ht. (cm) * Ht. (cm)) and place the results in column D, starting from D2 to D7. Ensure that the header 'BMI' is added in cell D1. Then, create a scatter chart with a line marker subtype, using 'Age' from A2:A7 as the x-axis and the calculated 'BMI' from D2:D7 as the y-axis. Title the chart 'Age' and label the axes as 'Age' and 'BMI'.
2	In Sheet1, compute the BMI for each row using the formula 10000 * Wt. (kg) / (Ht. (cm) * Ht. (cm)) and add the results in column D, with 'BMI' as the header. Then, create a scatter chart with 'Age' as the x-axis and 'BMI' as the y-axis, and title it 'Age'.
3	Calculate BMI for each person in Sheet1 using their height and weight, and place the results in column D. Add a header 'BMI' in D1. Create a scatter chart with 'Age' as the x-axis and the calculated 'BMI' as the y-axis, titled 'Age'.
4	Add a BMI column in Sheet1 using height and weight data, and create a scatter chart with Age and BMI.
5	Create a scatter chart in Sheet1 using Age and BMI data.



(a) Distribution of number of trajectories for each workbook



(b) Composition of various artifacts



(c) Variation in trajectory length for each artifact type

Figure 2: Statistics on distribution within the dataset.

- The single-round setting used to evaluate **SpreadsheetBench** (SB) which uses `openpyxl` to interact with spreadsheets [5].
- A **VBA** baseline that SheetCopilot was compared against. Besides providing only known tables, as done by SheetCopilot where all data starts in cell `A1`, we additionally provide all cell values in a Markdown format, trading off cost for performance.

All evaluations are carried out against OpenAI `gpt-4o` at temperature 0.

4.1 Metrics

We present a heuristic evaluation that verifies if the intended artifact was generated. For charts, we verify the type and the data that it points to. For pivot tables, we verify the data used and the values (accepting differences in order). For conditional formatting, we verify if the right cells are colored (but ignore the color itself). For formulas, we verify if the newly added cell range matches by value (execution match).

5 Results and Analysis

We analyze the performance for increasing abstractness of the instruction, increasing the number of steps in the user trajectory, and the performance on creating different artifacts.

5.1 Abstractness of instruction

Table 3 shows the success rate for increasing the level of abstractness of the instruction. The VBA baseline consistently outperforms the specialized approaches, even without the full sheet information. This is due to (1) these approaches overfitting on complex problems that require multiple steps, and (2) these approaches not being able to properly generate charts and pivot tables because of `openpyxl` limitations. This is further highlighted in Sections 5.2 and 5.3, respectively.

We further break down the failure cases into providing a wrong solution or providing no solution at all. VBA is harder to generate than `openpyxl`, which causes a different distribution of failure modes. The iterative refinement step of SheetCopilot causes very few instances where no candidate solution is provided. Adding full sheet context allows the model to write better VBA, because it has access to all range information.

5.2 Number of steps

Table 4 shows the performance for increasing the number of steps in each user trajectory for instructions level 1 and 5. Problems with fewer steps tend to require chart or pivot table generation, which the agentic systems using `openpyxl` struggle to generate due to API limitations. When the instruction is more abstract, simple baseline stops performing

Table 3: Performance for different approaches on different levels of instructions.

Model	Instruction level 1	2	3	4	5
Success					
VBA	64.50	55.73	46.95	42.37	41.22
+ MD	**71.37**	**62.21**	**64.89**	**51.51**	**42.37**
SheetCopilot	51.91	37.79	35.11	30.53	32.82
SheetAgent	38.17	28.24	29.77	18.70	22.52
SB	34.35	26.34	22.90	24.05	28.24
Wrong solution					
VBA	**20.99**	**25.19**	**31.30**	**29.77**	**40.84**
+ MD	22.90	27.86	27.10	38.17	45.80
SheetCopilot	45.80	60.31	60.69	69.08	66.41
SheetAgent	53.05	57.63	57.25	74.05	67.94
SB	47.71	49.24	51.15	50.38	49.24
No solution					
VBA	14.51	19.08	21.75	27.86	17.94
+ MD	5.73	9.93	8.01	10.32	11.83
SheetCopilot	**2.29**	**1.90**	**4.20**	**0.39**	**0.77**
SheetAgent	8.78	14.13	12.98	7.25	9.54
SB	17.34	24.43	25.95	25.57	22.52

well, but the agent is able to break down the task into different steps and execute on them.

Table 4: Pass rate per complexity against number of steps for the trajectories for utterance levels 1 and 5.

Model	# Steps 1	2	3−4	5+
Level 1				
VBA baseline	81.40	58.14	53.85	35.29
+ MD	**89.15**	65.12	46.15	**50.98**
SheetCopilot	43.41	**79.07**	**61.54**	43.14
SheetAgent	43.41	25.58	35.90	37.25
SB	40.31	20.93	35.90	29.41
Level 5				
VBA baseline	66.67	27.91	20.51	3.92
+ MD	**67.44**	27.91	17.95	9.80
SheetCopilot	36.43	**41.86**	**33.33**	15.69
SheetAgent	18.60	16.28	**33.33**	29.41
SB	30.23	20.93	25.64	**31.37**

5.3 Artifact type

Table 5 shows the performance for different types of objects. SheetAgent and SpreadsheetBench use `openpyxl`, which has no support for inserting charts and limited support for inserting pivot tables. Interestingly, SheetAgent and SpreadsheetBench overfit on detailed instructions (level 1) and do not leverage the provided context, causing them to fail on problems that require pointing to specific ranges of data.

Table 5: Pass rate per complexity for each object type across utterance levels 1 and 5.

Model	Artifact type Formulas	Charts	PT	CF
Level 1				
VBA baseline	40.50	87.88	**100.00**	**74.29**
+ MD	55.37	**90.91**	57.14	**74.29**
SheetCopilot	60.33	35.35	71.43	65.71
SheetAgent	**61.16**	0.00	0.00	**74.29**
SB	52.89	0.00	0.00	**74.29**
Level 5				
VBA baseline	29.75	58.59	**57.14**	28.57
+ MD	33.06	**59.60**	28.57	28.57
SheetCopilot	30.58	33.33	28.57	**40.00**
SheetAgent	38.84	2.02	14.29	25.71
SB	**50.41**	1.01	28.57	28.57

6 Conclusion

We propose the *workbook time machine* that generates benchmarks of spreadsheet object creation from final spreadsheets. We analyze the performance of existing agents and baselines on the benchmarks across three dimensions that we can finely control: task complexity as number of steps, the level of abstraction of the instruction, and the type of object that needs to be generated. This evaluation highlights limitations of spreadsheet manipulation agents: the choice of API and the level of detail in the utterance.

7 Limitations

Our benchmark is limited to the creation of derived objects and not their modification or deletion. Derivable cell values that are stored without formula, like the capital cities of countries, are not considered. The formula descriptor removal step can be adapted to consider these.

When removing formulas from ranges that are not stored as tables, other cells are not moved, which can cause some structural information to leak. Our evaluation does not consider the exact position of formulas, limiting the scope of the leakage to hints that *any formula will be needed*. This can be further resolved by a manual cleanup pass over the generated workbooks, which can then be used to evaluate automated workbook cleaning.

The current version of the benchmark is restricted to English utterances. The pipeline itself is language-agnostic and can be adapted to generate benchmarks across different languages.

References

[1] Titus Barik, Kevin Lubick, Justin Smith, John Slankas, and Emerson Murphy-Hill. 2015. Fuse: a reproducible, extendable, internet-scale corpus of spreadsheets. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 486–489. IEEE.

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

[3] Felienne Hermans and Emerson Murphy-Hill. 2015. Enron's spreadsheets and related emails: A dataset and analysis. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 7–16. IEEE.

[4] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and ZHAO-XIANG ZHANG. 2023. Sheetcopilot: Bringing software productivity to the next level through large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 4952–4984. Curran Associates, Inc.

[5] Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908.

[6] Justin Payan, Swaroop Mishra, Mukul Singh, Carina Negreanu, Christian Poelitz, Chitta Baral, Subhro Roy, Rasika Chakravarthy, Benjamin Van Durme, and Elnaz Nouri. 2023. Instructexcel: A benchmark for natural language instruction in excel. *arXiv preprint arXiv:2310.14495*.