

Project : Minerva_NLQA4Spreadsheet

Title : Natural Language-based Question Answering from Multiple Spreadsheets via RAG and code-based LLM Agents

Abstract

In the era of LLM-based generative AI, natural language has become the most common way to interact with any types of text document. By uploading the document and asking question about the document to an LLM, an user can retrieve desired information about the document. Although asking question via prompts help to retrieve information from unstructured documents like pdfs, LLMs often struggle answering natural language queries for structured documents e.g. spreadsheets. Firstly, the context window of LLMs are limited, therefore for scenarios with large number of rows in a spreadsheet, it is not possible to feed an entire spreadsheet into the context window of LLM. Secondly, unlike unstructured document like pdfs, spreadsheet often can be represented as tabular layout structure which is ideal for extracting necessary information by writing a code, rather than directly providing the table like structure to LLM's prompt which may increase LLM hallucination. Thirdly, when natural language query is from multiple spreadsheets, it is time consuming and require manual effort to fetch the correct column names from metadata containing information about different spreadsheets and combine the information. Moreover, when the query is complex and composed of several subqueries, it is necessary to breakdown the query into several subqueries, process each of the subquery separately and combine them to finally answer the full query. To meet these challenges, we introduce two different retrieval methods by matching the query-spreadsheet description vector and subquery-column description vector to retrieve necessary spreadsheet metadata to answer multiple spreadsheet-based query from a set of spreadsheets. After retrieving relevant spreadsheet metadata via top-K retrieval with matching between query-spreadsheet or subquery-column combination, we provide this metadata to an LLM-Agent with which writes an appropriate code and executes it to find the necessary information from natural language query from multiple spreadsheets. We performed extensive experimentation for queries originating from single spreadsheet and multiple spreadsheets from a given set of structured spreadsheet in proper tabular format. For queries originating from single spreadsheet, we evaluate on five datasets of the databench benchmark and present the accuracy and per query running time. Also for queries originating from multiple spreadsheets, we prepared 25 questions where each queries are created combining subqueries from 2-3 datasets of databench benchmark. In this dataset, we have observed that providing metadata about all spreadsheet achieves a 72% accuracy, query-spreadsheet matching and subquery-column matching achieves 84% and 80% accuracy respectively. We also provide per query running time comparison between the all metadata and two retrieval approaches.

Problem Definition: Given a set of N spreadsheets in .csv format, and query Q in natural language from an user based on the given set of spreadsheets. The problem is to find the answer of this query within a limited time limit t. In this project, we assume that each spreadsheet denotes a table where each row represents a record and each column represent different information of the record

Natural Language Question Answering for Spreadsheets

Project Minerva: For the ongoing Minerva project at Amazon, a user can upload a document which only allows unstructured document e.g. pdf files or doc files or text files and ask a natural language query on the uploaded document. LLMs with RAG used to select the relevant uploaded document to answer the query and it will cite the source of the answer. However, for

structured documents e.g. excel files or spreadsheets it is often difficult to fetch the answer of a question by only reading the file since LLMs are not very good dealing with tabular files. Therefore, in this project we aim at answering natural language queries from spreadsheets which is not currently covered by the Minerva project.

Challenges of natural language question answering from multiple spreadsheets:

- LLMs not suited for extracting information from structured documents: Unlike unstructured documents e.g. pdfs and text files, spreadsheets contain data in tabular format where the column names contain semantic information about the column values in different rows. LLMs are good at semantic similarity based search for unstructured data and not ideal to extract information for tabular data in spreadsheet as they suffer from hallucination problem while searching over spreadsheets and combining the information to answer a natural language query. When the number of columns and the number of rows in spreadsheet increases, LLM struggles with the lost in the middle problem to search on those large spreadsheets. To address with the issue of retrieving the necessary information from spreadsheet, writing python code is a viable approach. By extracting the necessary information from an spreadsheet via a python code, the provided natural language query can be easily answered and this approach is more scalable for larger spreadsheets and more accurate as well. Therefore, an approach to make LLM write a python code which extract the necessary information from spreadsheet and answer the query with the extracted information.
- Difficulty in identification of correct column name extraction: Even with the code writing approach to answer natural language query, there exist the problem of correct column names extraction from the spreadsheet metadata. Complex queries are made up of several simple subqueries. for spreadsheet with large number of columns it becomes difficult for LLMs to accurately retrieve the correct column names necessary for each subquery. Also, when the query is from multiple spreadsheets and the number of spreadsheet is large LLMs often struggle to find the necessary spreadsheets to answer the query. If we provide the metadata of all the spreadsheets to the code writing agent, in that case the task of necessary column extraction to write python code to answer the query become even more difficult for LLMs. Hence, it is important to extract a small subset of spreadsheets which contains columns necessary to answer a spreadsheet query.

Proposed solutions:

- Code generation and execution for NLQA on spreadsheet: Natural Language Query from spreadsheet with LLMs is difficult compared unstructured document therefore for structured document like spreadsheet it is often better to make LLM write a code and execute that to extract relevant information
- Relevant metadata extraction via query-spreadsheet similarity matching: To write code for natural language query on spreadsheet, it is important to identify the correct column names from the spreadsheet metadata (spreadsheet name, spreadsheet description, column name, column description, column values). But for large number of spreadsheets and each spreadsheet having more columns, it is often difficult to identify correct column names for writing the appropriate code. To resolve this issue, we generate query description and spreadsheet description and convert them into vector and perform query-spreadsheet matching to extract the relevant spreadsheet necessary for answering the query.
- Granular level subquery-column matching for relevant metadata extraction: While similarity matching between query and spreadsheet might extract relevant spreadsheets it still adds extra information in the spreadsheet description. Often queries are composed of multiple subqueries for this it is better to break them into subqueries, solve each of them and combine the result of each query to answer the entire subquery. To perform similarity matching at granular level, we breakdown the query into subqueries to such extent that it is sufficient to answer a subquery from a particular column. Next, we also describe each column with LLM so that it contain enough description to be matched by the necessary subquery. We convert both the subquery description and column description into vectors and perform similarity matching and then choose the relevant spreadsheets of top-K most similar columns with the subquery.

We describe two different process of relevant metadata extraction via query-spreadsheet matching or subquery-column matching in the following overview diagram:

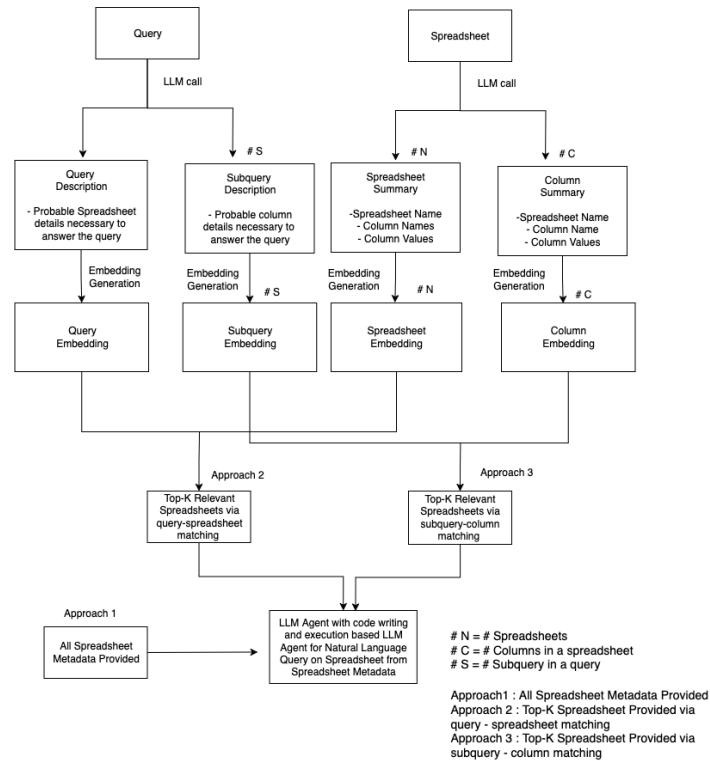


Figure 1: Overview Diagram of the proposed approaches with code writing and execution based LLM Agent for Natural Language Query from Multiple Spreadsheet: 1) all spreadsheet metadata provided, Top-K relevant spreadsheet provided via 2) query-spreadsheet matching 3) subquery-column matching

QUERY-SPREADSHEET MATCHING BASED RELEVANT SPREADSHEET RETRIEVAL NATURAL LANGUAGE QUESTION ANSWERING:

We will describe each step of the **query-spreadsheet matching** for relevant spreadsheet retrieval for natural language question answering on spreadsheet in the following sections with appropriate implementation details:

1. **Spreadsheet summary generation:** To perform similarity matching between the query description and spreadsheet description for relevant spreadsheet retrieval and put them into the metadata of code generation and execution agent, we first need to obtain the description of each spreadsheet consisting of the spreadsheet name, column name and a sample of column values. We experimented with the first five column value for each column which can be something different as well e.g. min, max, mode, number of categories etc for giving LLM an understanding about the types of column values which might help in code writing and execution.

We use the following prompt with a sample for providing the unformatted spreadsheet summary generation:

Prompt for Unformatted Spreadsheet summary Extraction:

```
"" You are an agent to provide paragraph summary of spreadsheets
given the metadata as spreadsheet names and column names.\n
You will receive spreadsheet metadata in the following formats.\n\n
spreadsheetname.csv
```

```
column1: value1a, value1b, value1c
column2: value2a, value2b, value2c
column3: value3a, value3b, value3c
\n\n
```

Your task is to produce a summary of the spreadsheet for each column name.
This summary will be used to search required column names to answer query.\n\n

STRICTLY follow the following format to generate summary WITHOUT ANY other output.

spreadsheetname.csv describes spreadsheet description,

containing column1 which describes column1 description
with values value1a, value1b, value1c, value1d, value1e,

containing column2 which describes column2 description
with values value2a, value2b, value2c, value2d, value2e,

containing column3 which describes column3 description
with values value3a, value3b, value3c, value3d, value3e.

The spreadsheet metadata is as follows:\n\n
""

Example of unformatted spreadsheet summary:

001_Forbes.csv describes Forbes billionaires data, containing rank which shows numerical ranking positions with values 1, 2, 3, 4, 5, containing personName which lists billionaire names with values Elon Musk, Jeff Bezos, Bernard Arnault & family, Bill Gates, Warren Buffett, containing age which shows the age of billionaires with values 50.0, 58.0, 73.0, 66.0, 91.0, containing finalWorth which shows net worth in millions with values 219000, 171000, 158000, 129000, 118000, containing category which indicates business sectors with values Automotive, Technology, Fashion & Retail, Technology, Finance & Investments, containing source which lists primary companies or sources of wealth with values Tesla, SpaceX, Amazon, LVMH, Microsoft, Berkshire Hathaway, containing country which shows residence countries with values United States, France, containing state which shows residence states with values Texas, Washington, Nebraska, containing city which shows residence cities with values Austin, Seattle, Paris, Medina, Omaha, containing organization which lists primary business organizations with values Tesla, Amazon, LVMH Moët Hennessy Louis Vuitton, Bill & Melinda Gates Foundation, Berkshire Hathaway, containing selfMade which indicates if wealth is self-made with values True, False, containing gender which shows gender with value M, containing birthDate which shows birth dates in datetime format, containing title which shows business positions with values CEO, Entrepreneur, Chairman and CEO, Cofounder, CEO, containing philanthropyScore which rates charitable giving with values 1.0, 4.0, 5.0, containing bio which provides detailed biographical information about the billionaires careers and achievements, containing about which provides additional personal details and interesting facts about the billionaires.

2. **Query description generation:** Just like the spreadsheet we also generate the summary of query using LLM produce query description by converting them into subqueries which will be more detailed than the actually query and ask LLM to predict possible column names to answer each subquery so that while performing similarity matching it can retrieve the spreadsheets with the correct column names.
3. **Top-K spreadsheet extraction via query-spreadsheet matching:** After generating the summary of subquery and spreadsheet we obtain the embedding from Titan model of both the query and spreadsheets. Next, we perform cosine similarity matching between the query embedding and the spreadsheet embeddings and choose the top-K similar spreadsheets for each query.

We describe the overview diagram of query-spreadsheet matching based relevant spreadsheet extraction for natural language query on the following diagram.

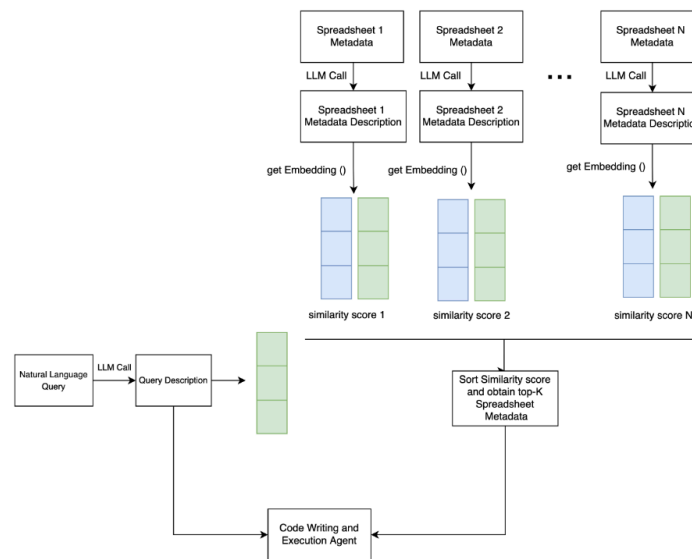


Figure 2: Query-Spreadsheet Description Matching RAG for Spreadsheet Extraction

The intuition behind this design choice is the fact that cosine similarity between two documents increases if the context of two different document is similar. When we describe a query into subqueries and ask LLM to provide details about probable column names necessary for each subquery and also produce the description of each spreadsheet with spreadsheet description, column description and column values there is a higher chance that those spreadsheets from which the query is asked will have a higher cosine similarity scores with the query embedding and as a result the necessary spreadsheets will be extracted.

SUBQUERY-COLUMN MATCHING BASED RAG FOR SPREADSHEET EXTRACTION

Secondly, we define another approach which we term as **subquery-column similarity matching for relevant spreadsheet metadata extraction**. We observe that often queries are composed as several subqueries where each subquery can be answered by performing some operation on a single column from some spreadsheet. This means if we can describe the query into simple subqueries so that we can perform similarity search on different column description. Therefore, we ask LLM to breakdown the query into subqueries and generate subquery description, which can be answered from a single column, and also ask LLM to provide column description providing spreadsheet name, column name and some values (say 5) of that column. After that, we convert subquery description into subquery embedding and column description into column embedding. Next, we compute the cosine similarity and pick the top-K columns and corresponding spreadsheet names necessary to

answer each subquery. We provide the metadata of the extracted spreadsheet names to the code writing agent and ask it to write python code to answer the query.

Cosine Similarity-based Vector Search for spreadsheet metadata extraction : Transformer based LLMs provide embeddings of the input text in the last layer which contains semantics about the input text. Cosine similarity is defined as the dot product between two normalized vectors which compute the cosine angle between the two vectors in the latent space. A higher cosine similarity denotes two vectors are closer in the latent space. Therefore, given a query vector q and a set of documents D , the documents which are more contextually similar with the context of the query will have a higher similarity in their corresponding vector representation. We utilize this idea to fetch the metadata of the most relevant spreadsheet from a list of spreadsheets. From our experiments we have observed that when we provide the metadata about all available spreadsheets then LLM struggle to find out the relevant column name. We have also observed that if we only include metadata about the relevant spreadsheet or a few spreadsheet e.g. 5 instead of metadata from all spreadsheet in that case LLMs can identify the column name and write a python code to answer the query. Observing this we first breakdown the query into a subquery with an LLM call and convert each subquery into an embedding. Also, we feed spreadsheet name, column names and column values to LLM and obtain column level spreadsheet metadata with details about the spreadsheet, column and values and convert each column metadata into an embedding. Next, for each subquery we perform vector similarity search between the subquery embeddings and the embedding of column level spreadsheet metadata. For each subquery we choose the top-K similar spreadsheets and combine the extracted spreadsheet in a set. We provide the metadata of the extracted spreadsheet set to the code writing agent and execute a python code to answer the natural language query.

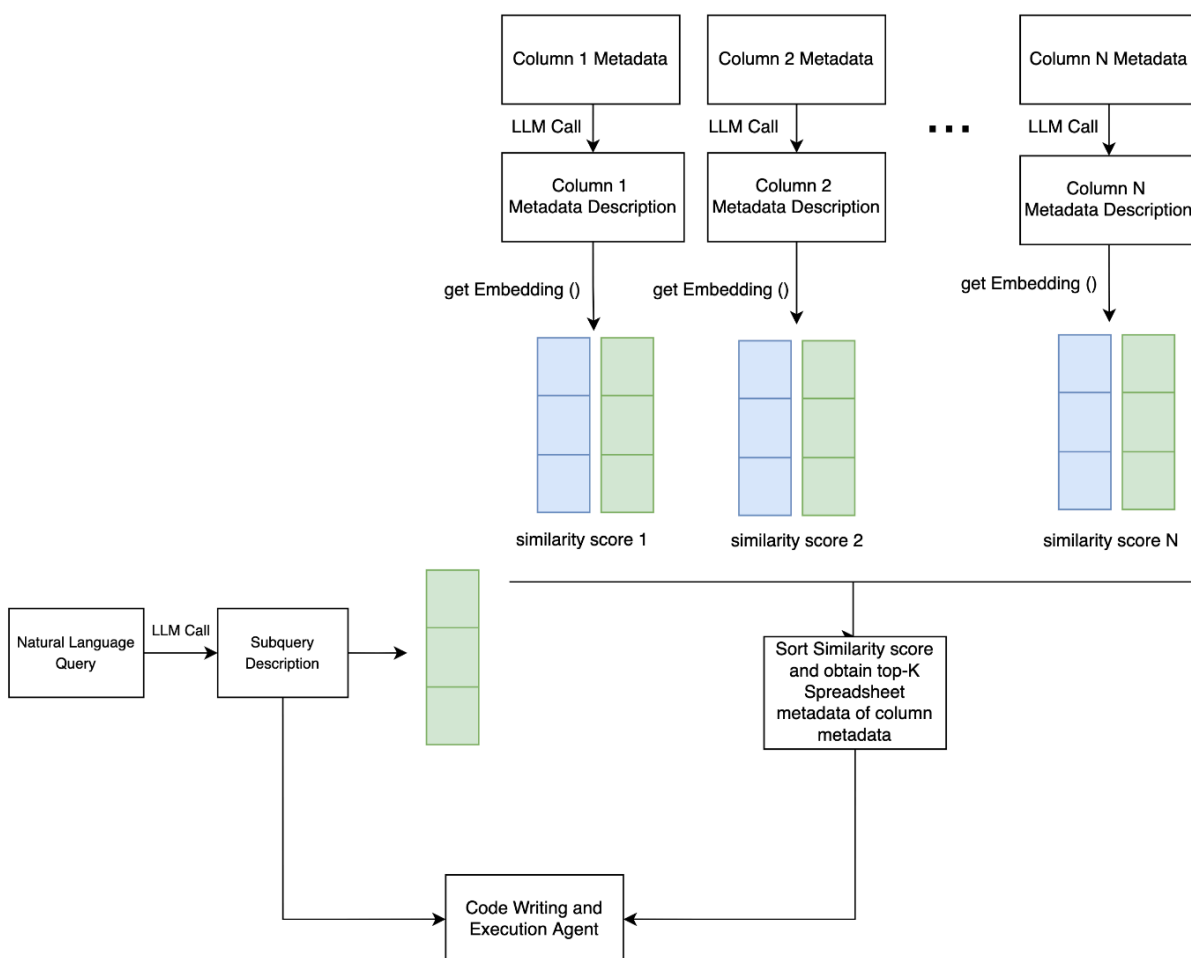


Figure 3: Metadata Extraction Pipeline via Subquery-Column Description Matching

Code generation and execution based LLM Agent: In this step we put the formatted summary of extracted top-K spreadsheet metadata into the prompt of LLM agent which writes a python code and execute it to find the answer of each query utilizing the necessary column names to answer the query in python code. This step is common in both approach and the presence of the spreadsheets from which the query is actually from is necessary to correctly answer the query in natural language.

EXPERIMENTAL RESULTS

Experimental Setup: To evaluate the efficacy of query-spreadsheet and subquery-column matching based relevant metadata extraction methods over providing all metadata to the code writing and execution based LLM Agent for natural language question answering, we performed experimental analysis in two different setting. We use sonnet-v3.5 as the LLM from Anthropic and Titan-embed-text-v1 from Amazon as the embedding generation model. For the code generation and execution we used strands api [16] which enables the use of tools e.g. with LLMs.

For queries from a single spreadsheet, we run all three approaches on a particular dataset from databench benchmark and report the running time and accuracy. Each of the dataset from databench benchmark contains around 25 questions solely from that dataset. For the task of question answering from single spreadsheet, the relevant spreadsheet metadata extraction approaches require searching over all the spreadsheets and the spreadsheet from which the query is actually from needs to be included the extracted spreadsheets to answer the question correctly.

Performance comparison to answer single spreadsheet-based questions from multiple spreadsheets: In Figure 3, we present the performance analysis on five different dataset from databench dataset (001_Forbes.csv, 002_Titanic.csv, 015_Food.csv, 026_Predicting.csv and 027_Supermarket.csv). The databench dataset contains around 65 spreadsheets and the question on those specific dataset are only from the corresponding single datasets. In our experimental setup, we present three different approaches. First, we experimented with providing metadata of all 65 spreadsheets without performing any top-K retrieval represented by the green bars in Figure 3. Next, we perform query-spreadsheet and subquery-column matching to retrieve top-K relevant spreadsheet necessary to answer the query. We can observe that the subquery-column matching approach roughly performs better than query-spreadsheet matching and all-metadata searching approach on 015_Food.csv, 026_Predicting.csv and 027_Supermarket.csv datasets while for the 001_Forbes.csv and 002_Titanic.csv dataset the performance difference between all three approaches is quite similar.

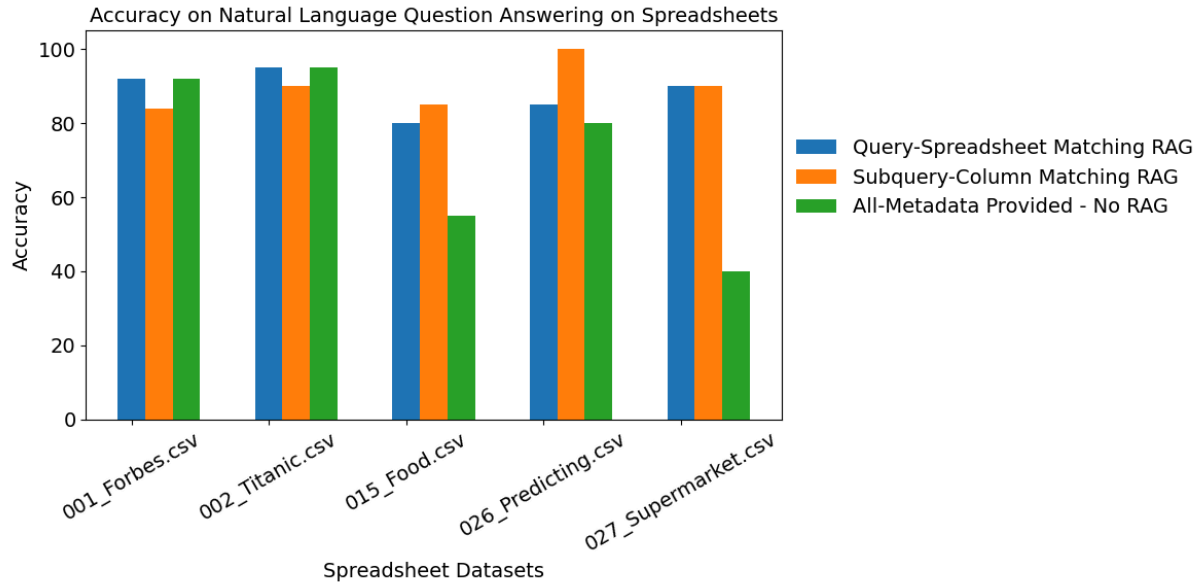


Figure 4: Performance analysis comparison of subquery-column matching RAG, query-spreadsheet matching RAG and all metadata without RAG approaches for query from single spreadsheet among multiple spreadsheets from databench dataset

Running time analysis to answer single spreadsheet-based questions from multiple spreadsheets: In Figure 4, we present the per query running time analysis for all three methods for different datasets as barcharts. From the bar chart, we can observe that the subquery-column description matching approach takes more time compared to query-spreadsheet matching approach and the approach with all metadata for 002_Titanic.csv, 026_Predicting.csv dataset while the query-spreadsheet-matching based approach takes comparatively more time for 001_Forbes.csv, 015_Food.csv and 027_Supermarket.csv dataset. The operations of converting the query/subquery into vectors and performing the cosine similarity based top-K retrieval with each of the spreadsheet/column description vector makes the approach slower than the all-metadata provided approach. The all-metadata provided approach is faster than the query-subquery/spreadsheet-column matching approach since there is no retrieval of relevant spreadsheets necessary to answer the query. query-subquery/spreadsheet-colu

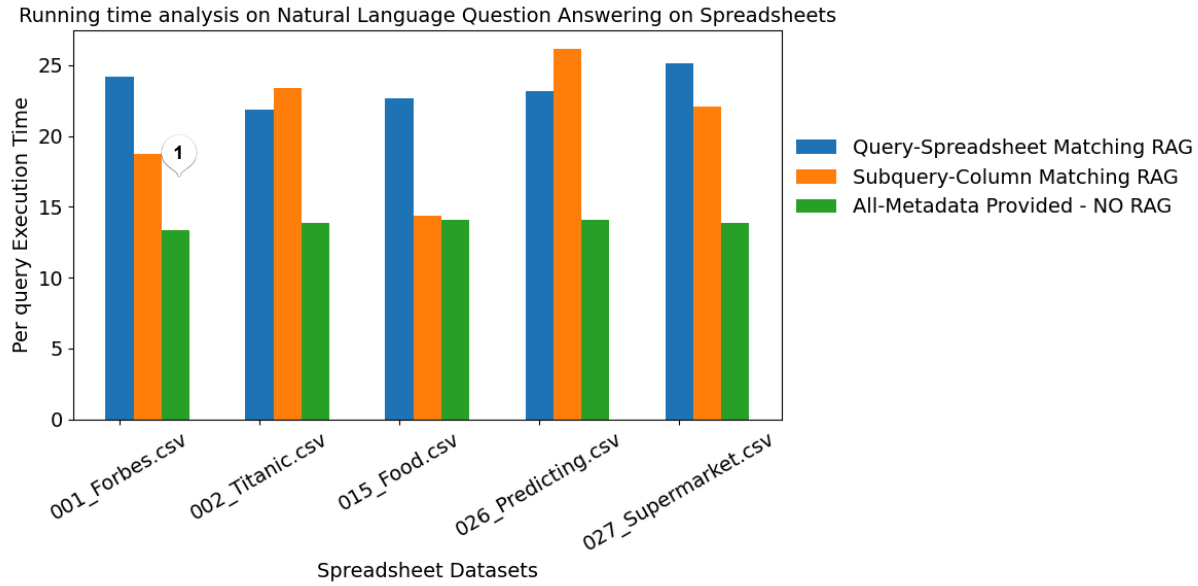


Figure 5: Running time analysis comparison of subquery-column matching RAG, query-spreadsheet matching RAG and all metadata without RAG approaches for query from single spreadsheet among multiple spreadsheets from databench dataset

For queries composed from multiple spreadsheets, we prepared a new dataset with 25 questions where questions are created from multiple spreadsheets from databench benchmark. We benchmarked all three approaches on the newly prepared dataset and report the result [link]. For the task of question answering from multiple spreadsheet, the expected spreadsheet needs to be included in the extracted spreadsheets to correctly answer the query. To retrieve relevant spreadsheets, we experimented with 5 different k values for top-K spreadsheet retrieval, where k could

For queries composed from multiple spreadsheets, we prepared a new dataset with 25 questions where questions are created from multiple spreadsheets from databench benchmark. We benchmarked all three approaches on the newly prepared dataset and report the result [link]. For the task of question answering from multiple spreadsheet, the expected spreadsheet needs to be included in the extracted spreadsheets to correctly answer the query.

We present the list of 25 questions from multiple spreadsheets below:

Are the youngest passenger to survive and the youngest billionaire has the same sex?

List the name of the passengers who survived and older than the youngest billionaire.

Mention the age of youngest lover to have PhD and the youngest male passenger to survive.

Are the customer with the lowest credit limit, the youngest billionaire and the youngest passenger are from the same sex?

How many male customers with a post-graduate degree has a lower age than the average age of a loan applicant who has an installment rate more than 2%?

What is the most common resident category of a patient who has a glucose level higher than 50 and the average age higher than the average age of a loan applicant who has an installment rate more than 2%?

Who are youngest three billionaires who are older than the average age of an patient who has a stroke?

Mention the three countries who has most billionaires younger than the average age of a loan applicant who has an installment rate more than 2%?

Mention the name of billionaire who has the same sex as the youngest Post-graduate customer.

Mention the name of the oldest billionaire from United States who has a smaller age than the oldest post-graduate male customer.

How many billionaire from United States has the same gender as the customer with the highest credit limit and younger than the oldest surviving passenger.

What is the name of the youngest surviving passenger who gave fare less than the smallest credit limit and have the same

sex as the youngest billionaire?

How many surviving passengers gave fare less than the smallest credit limit and have the same sex as the youngest billionaire?

Does the youngest billionaire has an age between the youngest passenger to survive and the oldest male customer who has a Post-graduate level education?

Mention the name of the youngest billionaire who has an age between the youngest passenger to survive and the oldest male customer who has a Post-graduate level education?

Mention the customerID of the youngest salaried customer who has the same sex as the youngest billionaire and older than the youngest passenger to survive.

How many salaried customers has the same sex as the youngest billionaire and older than the youngest passenger to survive.

Mention the name of the youngest billionaire from Tesla who is younger than the oldest salaried customer and have the same sex as the oldest survivor?

How many billionaires are younger than the oldest salaried customer and have the same sex as the oldest survivor?

Who are top five youngest billionaires younger than the oldest salaried customer and have the same sex as the oldest survivor?

How many billionaires are older than the person than the most number of pregnancies and have the same sex as the oldest survivor?

Mention the names of the 5 billionaires in the alphabetically sorted order who are older than the person than the most number of pregnancies and have the same sex as the oldest survivor?

Does the youngest billionaire, the youngest passenger to survive and the youngest person to have a happy moment have the same sex?

Mention the name of the gold winning athlete who has the same sex as the youngest billionaire and the oldest passenger to survive.

Mention the first five surviving passengers in alphabetical order who are older the youngest billionaire and have the same gender as the most gold medal winning gender from USA.

Performance analysis to answer multiple spreadsheet-based questions from multiple spreadsheets: In Figure 6, we present the performance of query-spreadsheet matching and subquery-column matching approaches over the number of retrieved spreadsheets (K) for answering natural language query from multiple spreadsheets on the dataset described above. We experimented with $k = 5, 10, 15, 20, 25$ and presented the performance of query-spreadsheet and subquery-column matching relevant spreadsheet retrieval approaches.

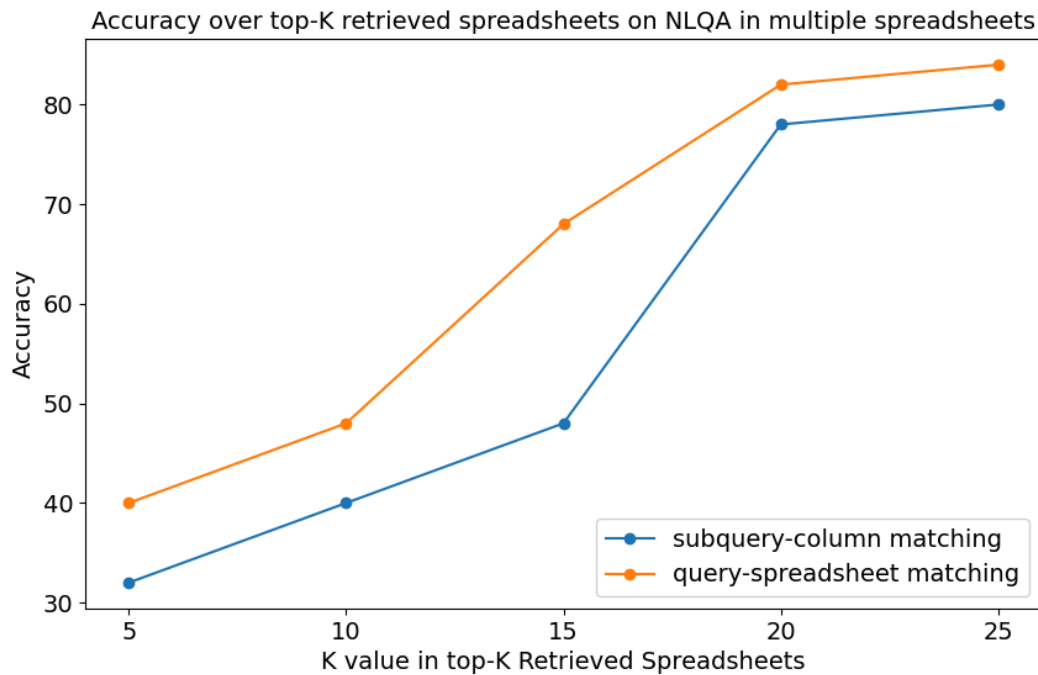


Figure 6: Performance analysis comparison over number of retrived spreadsheets (k) for subquery-column matching and query-spreadsheet matching approaches for query from multiple spreadsheet among multiple spreadsheets from databench dataset

We can observe that as the k approaches close to 20 the performance of both the approach almost converges. With increasing number of k we are allowing more number of spreadsheet to be extracted which might contain the two spreadsheets from which the questions are originally from and this leads to better accuracy. While for smaller number of k the relevant two spreadsheets from which the questions are originally from might not be present in the retrieved spreadsheet which leads to a accuracy drop for both approaches. We can observe that as we increase the number of spreadsheets (K) to be extracted for top-K relevant spreadsheets, the performance of both the query-spreadsheet matching and subquery-column matching aporoach increases. Allowing higher number of spreadsheets to be included in the spreadsheet metadata, increases the chance of expected spreadsheet to be included in the extracted spreadsheets. As a result, there is a sharp increase in the accuracy increases. However, we can observe little improvement in the accuracy between k=20 and k=25. Therefore, we can conclude that with k=20, there is a higher chances the extracted spreadsheets might include the expected spreadsheets from which the queries are actually from. It should be noted that originally the queries are prepared by combining informations from 2-3 spreadsheet from the databench benchmark. For the subquery-column matching we extract top-K relevant columns which can be from the same spreadsheet. Therefore, the number of extracted spreadsheet might very less for the subquery-column matching approach. Overall, we obtain the best performance when the number of retrieved spreadsheet is between k=20 to k=25 for both query-spreadsheet matching and subquery-column matching approach.

Running time analysis to answer multiple spreadsheet-based questions from multiple spreadsheets: In Figure 7, we present the performance of qurey-spreadsheet matching and subquery-column matching approaches over the number of retrived spreadsheets (K) for answering natural language query from multiple spreadsheets on the dataset described above. We experimented with k = 5, 10, 15, 20, 25 and presented the performance of query-spreadsheet and subquery-column matching relevant spreadsheet retrieval approaches.

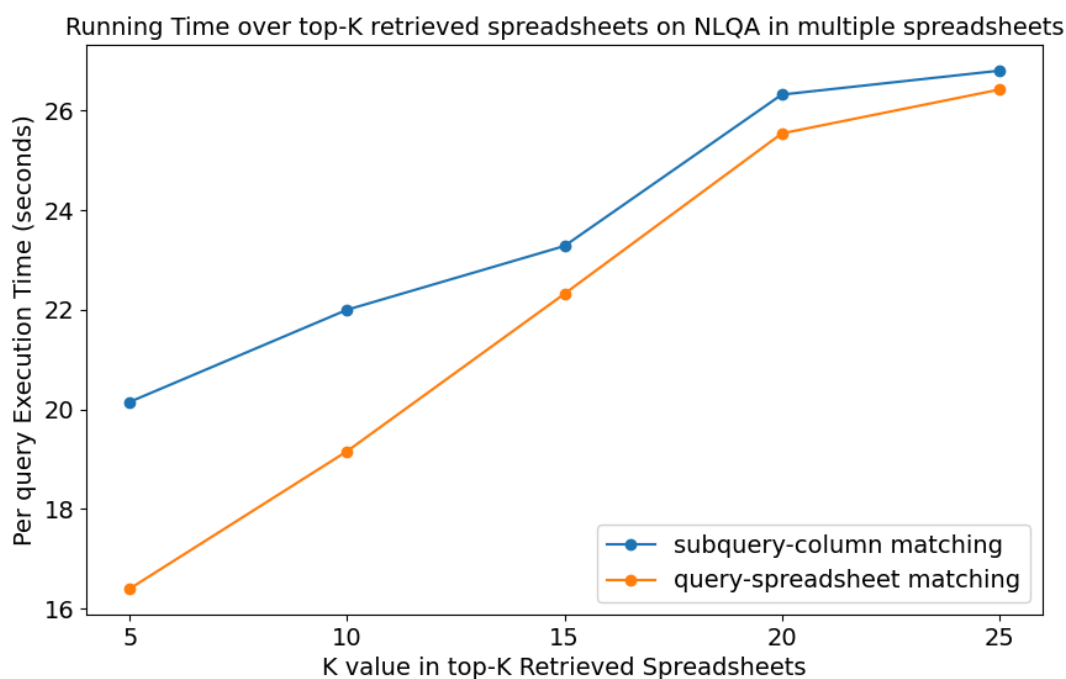


Figure 7: Running time analysis comparison over number of retrived spreadsheets (k) for subquery-column matching and query-spreadsheet matching approaches for query from multiple spreadsheet among multiple spreadsheets from databench dataset

We can observe from the line curves of Figure 7 that as we increase the number of retrieved documents the per query running time of both approaches increases. When a large number of spreadsheets are allowed to retrieve and appended as spreadsheet metadata for the code writing and execution based LLM agent to answer the natural language query the amount of time to process each query increases. Also, we can observe that the subquery-column matching approach is comparatively slower than the query-spreadsheet matching approach. To breakdown the query into subqueries and also to search at the column level and not in the spreadsheet level takes significantly more operation which affects the latency of the per query execution. As a result, while both of the approach takes longer time when the number of retrieved spreadsheet increases, the subquery-column matching based approach is slightly slower than the query-spreadsheet matching based approach as it performs relevant spreadsheet retrieval by searching at more granular level.

Conclusion

In this project, we target the problem of natural language question answering from a set of multiple spreadsheets where questions can be composed from a single spreadsheet or multiple spreadsheet. We utilized code writing and execution based LLM Agent pipeline to extract necessary information from the spreadsheets to answer a natural language query. Rather than providing all metadata to the code writing and execution based LLM Agent, we propose two different approachch for relevant spreadsheet extraction via query-spreadsheet description matching and subquery-column description matching. For questions from single spreadsheets, we achieved around 86-90% accuracy which is same as the baseline models performance. For questions from multiple spreadsheets, we achieved 84% and 80% for query-spreadsheet matching and subquery-column matching spreadsheet extractions when the questions are prepared from 2-3 different spreadsheets. From our comparative analysis, we observed that including 20-25 spreadsheets as the metadata to the code writing and execution based LLM agent leads to superior performance for both query-spreadsheet and subquery-column matching based approach and the query-spreadsheet matching approach provides higher accuracy in comparatively less per query execution time.

Future Research Directions:

- Add Create Read Update Delete Undo/Redo operation on the uploaded spreadsheets/tables based on natural language instructions using python code written by LLMs
- Perform NLQA on spreadsheet with multiple tables in a single spreadsheet or over multiple sheets within a spreadsheet file
- Perform value summarization minimum, maximum, unique values, range, date format in metadata to provide a better understanding to LLMs while fetching relevant column/spreadsheet and while writing the code

References:

- [1] Question Answering over Tabular Data with DataBench: A Large-Scale Empirical Evaluation of LLMs.
Jorge Osés Grijalba, L. Alfonso Ureña-López, Eugenio Martínez Cámara, and Jose Camacho-Collados. LREC-COLING 2024
- [2] SheetAgent: Towards A Generalist Agent for Spreadsheet Reasoning and Manipulation via Large Language Models
Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, Jianye Hao, Hangyu Mao, Fuzheng Zhang, WWW, 2025
- [3] SpreadsheetBench: Towards Challenging Real World Spreadsheet Manipulation,
Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, Jie Tang, NeurIPS 2024
- [4] SpreadsheetLLM: 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, EMNLP 2024
- [5] SheetMind: An End-to-End LLM-Powered Multi-Agent Framework for Spreadsheet Automation,
Ruiyan Zhu, Xi Cheng, Ke Liu, Brian Zhu, Daniel Jin, Neeraj Parihar, Zhoutian Xu, Oliver Gao
- [6] Leveraging ChatGPT for Excel: How large language models are changing spreadsheet practices, Martin Broadhurst, 2024
- [7] Can an LLM find its way around a Spreadsheet? Cho Ting Lee 2024
- [8] Large Language Models for Spreadsheets: Benchmarking Progress and Evaluating Performance with FLARE
Simon Thorne, 2025
- [9] TableTalk: Scaffolding Spreadsheet Development with a Language Agent
Jenny T. Liang, Aayush Kumar, Yasharth Bajpai, Sumit Gulwani, Vu Le, Chris Parnin, Arjun Radhakrishna, Ashish Tiwari, Emerson Murphy-Hill, Guastavo Soares, 2025
- [10] Toward Real-World Table Agents: Capabilities, Workflows, and Design Principles for LLM-based Table Intelligence
Jiaming Tian, Liyao Li, Wentao Ye, Haobo Wang, Lingxin Wang, Lihua Yu, Zujie Ren, Gang Chen, Junbo Zhao, 2025
- [11] Retrieval Augmented Generation Evaluation in the Era of Large Language Models: A Comprehensive Survey,
Aoran Gan, Hao Yu, Kai Zhang, Qi Liu, Wenyu Yan, Zhenya Huang, Shiwei Tong, Guoping Hu, 2025
- [12] Toolformer: Language Models Can Teach Themselves to Use Tools,
Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, Thomas Scialom, Meta 2023
- [13] ReAct: Synergizing Reasoning and Acting in Language Models
Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao, 2022
- [14] Code as Policies: Language Model Programs for Embodied Control
Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, Andy Zeng, 2022

[15] PAL: Program-aided Language Models

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, Graham Neubig, 2022

[16] Strands SDK for LLM Agents development <https://strandsagents.com/latest/>

Appendix:

Formatted Spreadsheet Summary Generation Pipeline:

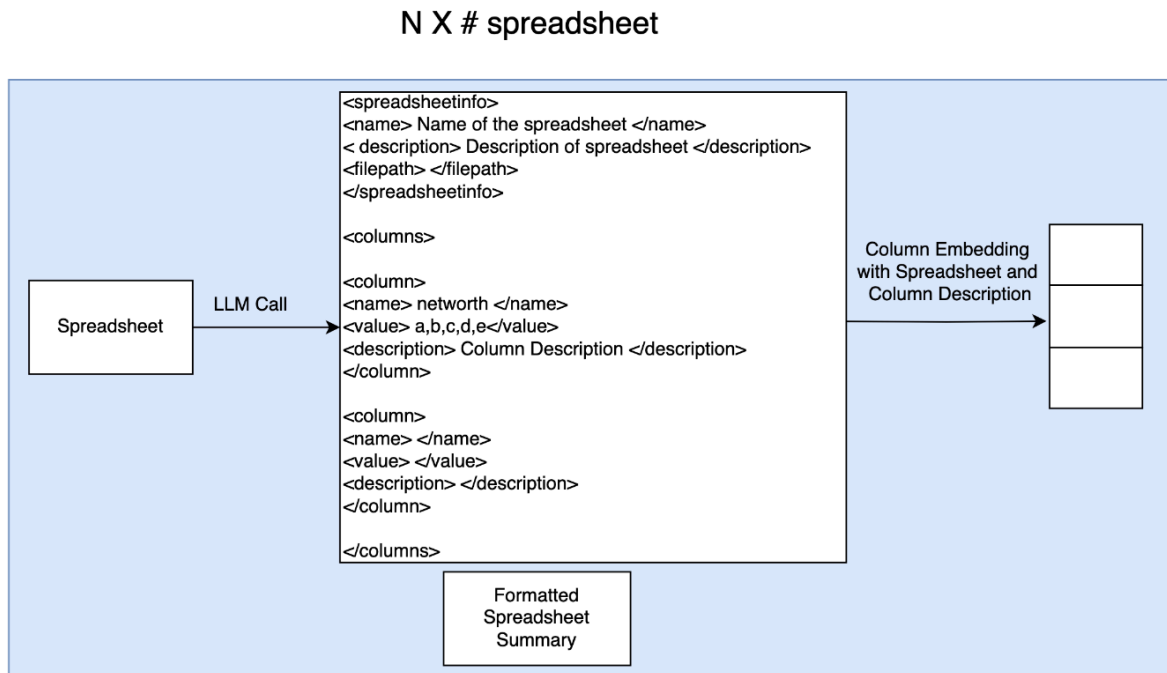


Figure : Formatted Spreadsheet Summary and Column Embedding Generation Pipeline

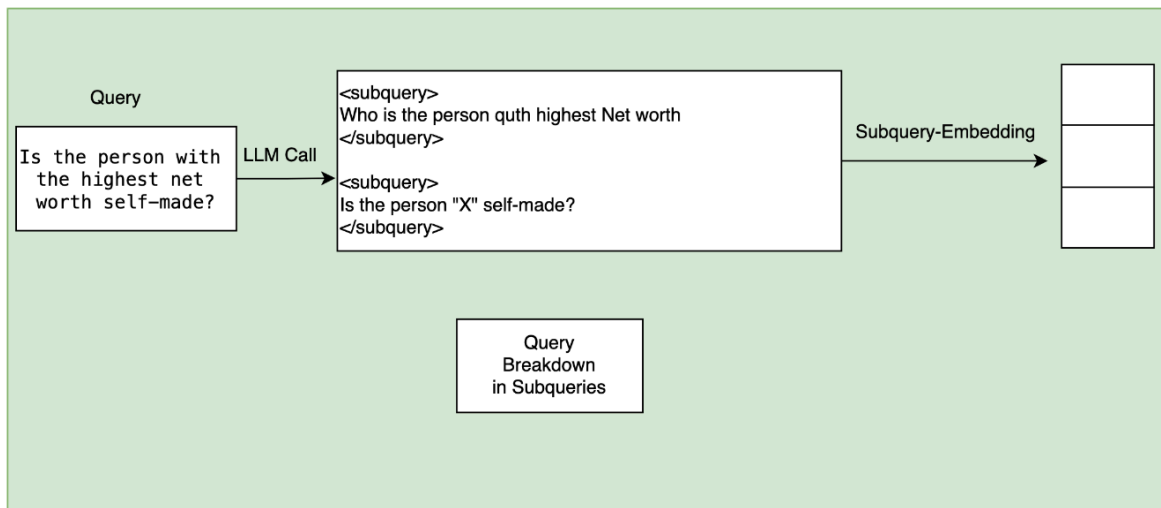


Figure: Query breakdown into subqueries via LLM call and subquery embedding generation

Prompt for Formatted Spreadsheet Summary Extraction:

```
system_command = "" You are an agent to provide paragraph summary of spreadsheets
given the metadata as spreadsheet names and column names.\n
You will receive spreadsheet metadata in the following formats.\n\n
spreadsheetname.csv
column1: value1a, value1b, value1c
column2: value2a, value2b, value2c
column3: value3a, value3b, value3c
column4\n\n
Your task is to produce a summary of the spreadsheet for each column name.
This summary will be used to search required column names to answer query.\n\n

STRICTLY follow the following format to generate summary.

<summary>
<spreadsheetname>spreadsheetname.csv</spreadsheetname>

<spreadsheetdescription> spreadsheet description </spreadsheetdescription>

<columns>

<column>
<name> column1 name </name>
<value> value1a, value1b, value1c </value>
<description> column1 description </description>
</column>

<column>
<name> column2 name </name>
<value> value2a, value2b, value2c </value>
```

```

<description> column2 description </description>
</column>

<column>
<name> column3 name </name>
<value> value3a, value2b, value3c </value>
<description> column3 description </description>
</column>

</columns>

</summary>

The spreadsheet metadata is as follows:\n\n
""

```

Example of Formatted Spreadsheet Summary:

```

<summary>

<spreadsheetname>001_Forbes.csv</spreadsheetname>
<spreadsheetdescription>This spreadsheet contains detailed information
about the top billionaires from Forbes, including their personal information,
wealth details, business interests, and biographical information.
</spreadsheetdescription>

<columns>

<column>
<name>rank</name>
<value>1,2,3,4,5</value>
<description>The ranking position of billionaires in the Forbes list</description>
</column>

<column>
<name>personName</name>
<value>Elon Musk,Jeff Bezos,Bernard Arnault & family,
Bill Gates,Warren Buffett</value>
<description>Full names of the billionaires</description>
</column>

<column>
<name>age</name>
<value>50.0,58.0,73.0,66.0,91.0</value>
<description>Current age of the billionaires</description>
</column>
<column>

<name>finalWorth</name>
<value>219000,171000,158000,129000,118000</value>
<description>Net worth of billionaires in millions of dollars</description>
</column>

<column>

```



```

<name>category</name>
<value>Automotive,Technology,Fashion & Retail,Technology,Finance & Investments</value>
<description>Primary industry or business sector of the
billionaire's wealth</description>
</column>

<column>
<name>source</name>
<value>Tesla, SpaceX,Amazon,LVMH,Microsoft,Berkshire Hathaway</value>
<description>Main companies or sources contributing to the
billionaire's wealth</description>
</column>

<column>
<name>country</name>
<value>United States,United States,France,United States,United States</value>
<description>Country of residence of the billionaire</description>
</column>

<column>
<name>state</name>
<value>Texas,Washington,nan,Washington,Nebraska</value>
<description>State of residence for billionaires (if applicable)</description>
</column>

<column>
<name>city</name>
<value>Austin,Seattle,Paris,Medina,Omaha</value>
<description>City of residence of the billionaire</description>
</column>

<column>
<name>organization</name>
<value>Tesla,Amazon,LVMH Moët Hennessy Louis Vuitton,Bill & Melinda Gates Foundation,
Berkshire Hathaway</value>
<description>Primary organization or company associated with
the billionaire</description>
</column>

<column>
<name>selfMade</name>
<value>True,True,False,True,True</value>
<description>Indicates if the billionaire is self-made or
inherited wealth</description>
</column>

<column>
<name>gender</name>
<value>M,M,M,M,M</value>
<description>Gender of the billionaire</description>
</column>

<column>
<name>birthDate</name>
<value>1971-06-28 00:00:00+00:00,1964-01-12 00:00:00+00:00,

```

```

1949-03-05 00:00:00+00:00,1955-10-28 00:00:00+00:00,1930-08-30 00:00:00+00:00</value>
<description>Date of birth of the billionaire</description>
</column>

<column>
<name>title</name>
<value>CEO,Entrepreneur,Chairman and CEO,Cofounder,CEO</value>
<description>Current professional title or position of the billionaire</description>
</column>

<column>
<name>philanthropyScore</name>
<value>1.0,1.0,nan,4.0,5.0</value>
<description>Score indicating the level of philanthropic activity
(scale of 1-5)</description>
</column>

<column>
<name>bio</name>
<value>Detailed biographical information about each billionaire's career
and achievements</value>
<description>Extended biographical information about the billionaire's
life, career, and achievements</description>
</column>

<column>
<name>about</name>
<value>Additional personal facts and interesting details about
the billionaires</value>
<description>Additional personal information and interesting
facts about the billionaire</description>
</column>
</columns>
</summary>

```

N X # spreadsheet

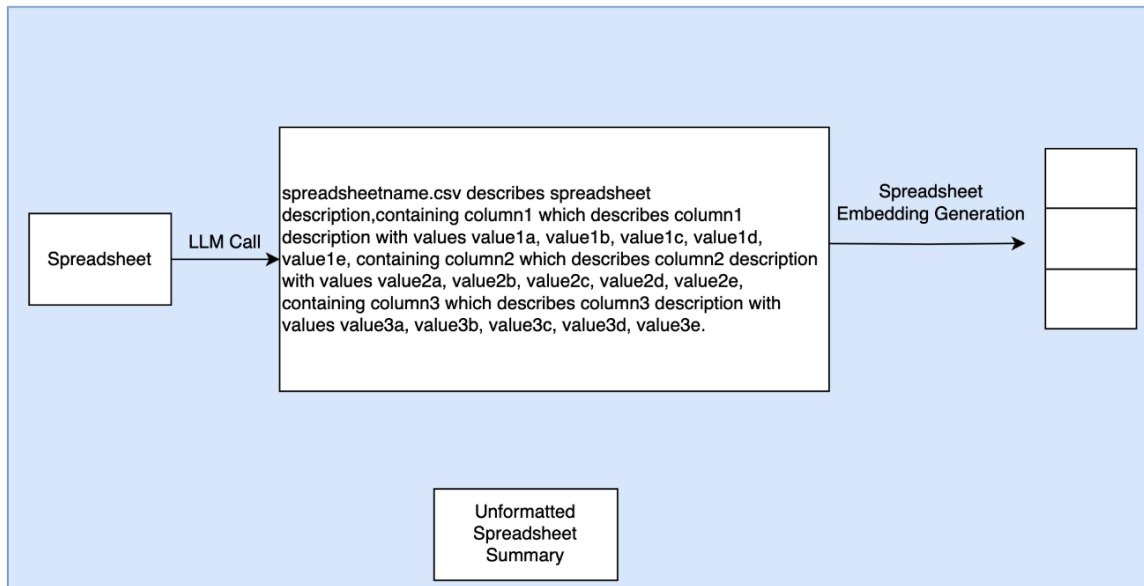


Figure: Formatted Spreadsheet Summary Generation and Spreadsheet Embedding Generation

Function for Query Breakdown into Subqueries with LLM call:

```
def breakdown_query(query):
    system_command = """ You are an query breakdown agent
    for spreadsheet-based natural language query.

    You will be provided a query within tag <query> </query>
    which you need to breakdown into subqueries at column level
    with enough details such that it can be used to search the
    corresponding column description to answer the subquery.

    Use the following format to break queries into subquery and
    return it inside <subquery> </subquery> tag, WITHOUT any other output.

    <subquery>The query query_provided_inside_query_tag contains subquery_description
    which require a column that describes column_description </subquery>

    <subquery>The query query_provided_inside_query_tag contains subquery_description
    which require a column that describes column_description </subquery>

    <subquery>The query query_provided_inside_query_tag contains subquery_description
    which require a column that describes column_description </subquery>

    Here is the query:
```

```

<query>"" + query + ""</query>""

query_breakdown_agent = Agent(model=bedrock_model)
output = query_breakdown_agent(system_command)
output = str(output)
return output

```

Sample Query Breakdown Example:

```

Query : Is the person with the highest net worth self-made?
From dataset : 001_Forbes
Breaking Down Into Subqueries for Spreadsheet Search:

<subquery>The query "Is the person with the highest net worth self-made?"
contains identifying people's net worth which require a column that describes
monetary value or net worth of individuals</subquery>

<subquery>The query "Is the person with the highest net worth self-made?"
contains determining if someone is self-made which require a column that
describes whether a person's wealth is self-made or inherited</subquery>

```

Prompt for NLQA4Spreadsheets via LLM Agents with code execution tool:

```

command = "" You are a natural language query assistant based on spreadsheets.

First, break down the query into subqueries with probable column name
necessary to answer them and store them in <subquery></subquery> tag.

Use the following format to break queries into subquery inside
<subquery> </subquery> tag.

<subquery>The query query_provided_inside_query_tag contains subquery_description
which require a column that describes column_description </subquery>

<subquery>The query query_provided_inside_query_tag contains subquery_description
which require a column that describes column_description </subquery>

<subquery>The query query_provided_inside_query_tag contains subquery_description
which require a column that describes column_description </subquery>

Next, you need to decide which column names are necessary to answer each of this
subquery. For this you are provided with the metadata describing spreadsheet summary
in following format. You have to find out which column names are necessary to answer
each of the subquery. If there are multiple possible candidate columns to answer a
subquery, choose the one which is more correct in terms of context.

<summary>
<spreadsheetname>spreadsheetname.csv</spreadsheetname>
<description> spreadsheet description </description>
<columns>

```

```

<column>
<name> column1 name </name>
<value> value1a, value1b, value1c </value>
<description> column1 description </description>
</column>

<column>
<name> column2 name </name>
<value> value2a, value2b, value2c </value>
<description> column2 description </description>
</column>

<column>
<name> column3 name </name>
<value> value3a, value2b, value3c </value>
<description> column3 description </description> </column>
</columns>
</summary>

```

Finally, you will write a python code to answer the query within `<code>` tag that will ONLY access the necessary columns, execute the code with the `execute_python_code` tool and use the output of the execution to answer the query within `<answer>` tag.

Important Rules

- Use the spreadsheet file name and column name exactly the are mentioned within the metadata.
- For boolean queries, answer True or False within `<answer>` tag
- For numeric/string queries, provide ONLY the answer within `<answer>` tag
- For list queries, provide the list in the format [a , b, c] within `<answer>` tag
- Don't use any columns which are not necessary to answer the query.

The query will be given within `<query>` tag and the metadata will be given within `<metadata>` tag.

```
<query> "" + query + "" </query> <metadata> "" + metadata + ""</metadata> ""
```

Execution Output:

Query : Is the person with the highest net worth self-made?
 From dataset : 001_Forbis
 Breaking Down Into Subqueries for Spreadsheet Search:

```
<subquery>The query "Is the person with the highest net worth self-made?"
contains identification of people's net worth
which require a column that describes monetary value or net worth of individuals
</subquery>
```

```
<subquery>The query "Is the person with the highest net worth self-made?"
contains information about whether someone is self-made
```

which require a column that describes if a person's wealth is self-made or inherited</subquery>

<subquery>The query "Is the person with the highest net worth self-made?" contains identification of people's net worth which require a column that describes monetary value or net worth of individuals</subquery>

<subquery>The query "Is the person with the highest net worth self-made?" contains information about whether someone is self-made which require a column that describes if a person's wealth is self-made or inherited</subquery>

Extracted Spreadsheets:

001_Forbes.csv

Answering query:

Let me break this down step by step:

1. First, let's break down the query into subqueries:

<subquery>

1. Find the person with the highest net worth
2. Check if that person is self-made

</subquery>

2. Looking at the metadata, we need these columns:

- 'finalWorth' to find the person with highest net worth
- 'selfMade' to check if they are self-made

3. Here's the Python code to answer this query:

<code>

```
import pandas as pd
```

```
# Read the CSV file
```

```
df = pd.read_csv('001_Forbes.csv')
```

```
# Get the row with highest net worth
```

```
highest_worth_person = df.loc[df['finalWorth'].idxmax()]
```

```
# Check if they are self-made
```

```
is_self_made = highest_worth_person['selfMade']
```

```
print(is_self_made)
```

</code>

Let me execute this code:

Tool #1: *execute_python_code*

True

<answer>True</answer>

The answer shows that yes, the person with the highest net worth (Elon Musk with \$219 billion) is indeed self-made.

<answer>True</answer>

The answer shows that yes, the person `with` the highest net worth (Elon Musk `with` \$219 billion) `is` indeed `self-made`.

Answer: `True`

Prediction: `True`

Dataset 001_Forbes, Total Processed: 1, Total Correct: 1, Accuracy : 100.0