Spring 2024 Capstone Project: Midterm Progress Report

March 9, 2024

Model Metadata Chatbot with GenAI

Members

Tianyue Cao (tc3334), Lu Liu (ll3721), Wanxin Luo (wl2930), Xinwei Qiao (xq2236), Yao Xie (yx2845) (In alphabetical order)

Mentors

Industry Mentor:
Sydney Son, Senior Associate, KPMG US
Thomas Covella, Manager, Data Scientist, KPMG US
Chengwei Wang, Senior Associate, KPMG US

Data Science Institute Mentor:
Sining Chen, Professor, Columbia University

Contents

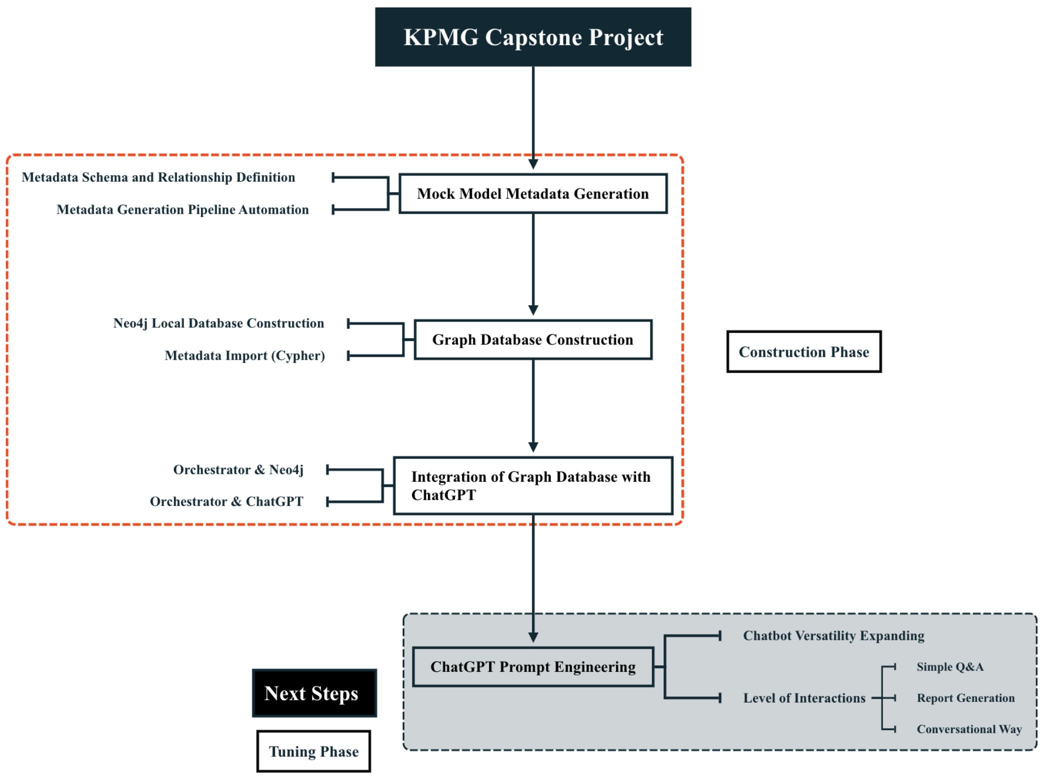
1 Introduction

1.1 Project Overview and Scope

The rapid evolution and integration of artificial intelligence and machine learning models into the core operations of businesses have brought about unprecedented efficiencies and innovations. However, this integration has also introduced complexities in managing the vast and interconnected landscape of these models. As models become more intertwined, with significant upstream and downstream impacts, the challenge of effectively managing model-related risks and ensuring comprehensive observability over these models intensifies. Recognizing the critical nature of these challenges, our capstone project, mentored by industry professionals from KPMG, aims to tackle these issues head-on by focusing on the management of model metadata.

This project sets out to develop a cutting-edge solution that leverages generative AI (genAI) technology to create a chatbot designed for seamless integration into day-to-day business operations. The core objective of this initiative is to enhance model risk management and increase the observability of models through the effective management of model metadata. By utilizing a back-end graph database to store the relational aspects of model interconnectedness, our solution aims to provide businesses with the ability to manage the complex landscape of their models more efficiently.

1.2 Project Roadmap



KPMG Capstone Project
Metadata Schema and Relationship Definition
Metadata Generation Pipeline Automation
Mock Model Metadata Generation
Neo4j Local Database Construction
Metadata Import (Cypher)
Graph Database Construction
Construction Phase
Orchestrator & Neo4j
Orchestrator & ChatGPT
Integration of Graph Database with ChatGPT
ChatGPT Prompt Engineering
Chatbot Versatility Expanding
Level of Interactions
Simple Q&A
Report Generation
Conversational Way
Next Steps
Tuning Phase

Figure 1: Project Roadmap

In the current phase of our project, we have completed the construction phase of this project, which includes:

1. Mock Model Metadata Generation
2. Local Graph Database Construction + Metadata Importation
3. Integration of Graph Database with GenAI Chatbot

We have successfully established a model entirely predicated on data generated in-house. This model is capable of addressing basic queries for specific use cases through prompt engineering, laying a solid foundation for enhancing its functionality and applicability in future stages. We have clarified the definitions of various components and their interrelations, which will serve as a basis for exploring broader applications of this model within the company's operations. This progress is a pivotal step towards leveraging our integrated system to improve efficiency and effectiveness in data management and user interactions, aligning with our strategic objectives.

2 Mock Model Metadata Generation

2.1 Metadata Schema and Relationship Definition

In the initial phase of our project, our primary task involves the generation of mock model metadata. This task is directly related to various data-centric work scenarios encountered at KPMG. Specifically, a data scientist at KPMG might be interested in understanding the performance of the models used in a specific report, the source of the data employed, which tables and even columns within the data warehouse were utilized, and the types of ETL (Extract, Transform, Load) transformations applied, etc. The data scientist might accordingly ask some questions to the GenAI chatbot and seek answers. To address these complex and common scenarios, we categorized the data to be generated into 5 distinct types. This systematic approach ensures that our mock model metadata generation is aligned with realistic work scenarios, thereby facilitating more meaningful analyses and insights.

Figure 2: Mock Data Types

Building upon five distinct data types, we have developed JSON-formatted schemas tailored to specific business scenarios for each category of data. For instance, in the case of data about reports, the schema typically encompasses attributes such as name, format, owner, content, and summary, among others. Figure 3 illustrates an example of a report content in JSON format, parsed according to our defined schema from a publicly available KPMG report. This approach enables us to systematically organize and interpret data, enhancing both its accessibility and utility in our ongoing project.

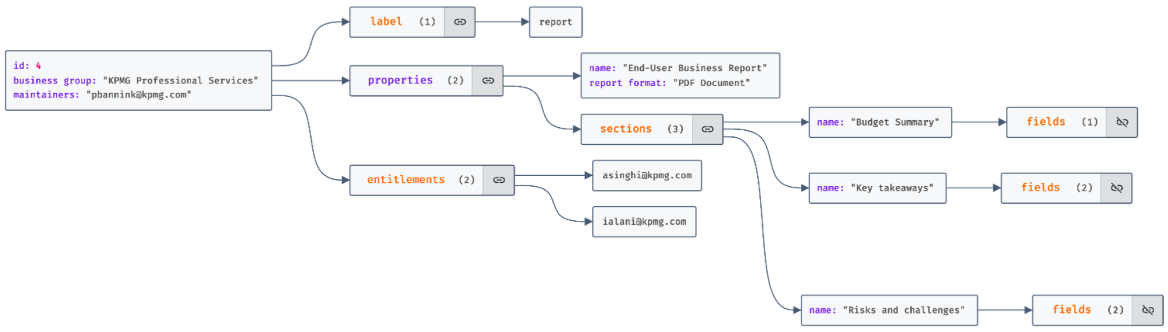


Figure 3: Report Example

To further our project's objectives, it is crucial to understand the relationships among different types of data to support more flexible use cases. For example, a report may include multiple models, with each model potentially evolving from iterations of previous models. The individuals responsible for maintaining or owning these models may vary, highlighting the need for a systematic approach to map and represent these entities and their connections, thus forming a knowledge network. To increase the system's flexibility, we have introduced instances specifically designed to map relationships between different entity types. These mapping instances aim to effectively capture and store the intricate web of relationships and additional details concerning these mappings. This methodical approach ensures a robust and interconnected data infrastructure, enabling deeper analysis and more meaningful insights of our chatbot.

In Figure 4, we present a diagram of the relationship network. This diagram centers on a report from KPMG official website and extends to illustrate the relationships between ML flows and data sources. These relationships are facilitated through two mapping instances. Additionally, we establish connections among ownership, data sources, and ML flows, as well as the internal components within data sources and ML flows, all through the addition of mapping instances. This results in the relational network depicted in Figure 4. Within such a network, when a user poses a question concerning a specific part of the report, a chatbot with access to this network has the capability to locate the instance where the answer resides, thereby providing the user with the requested information. This structure enhances the chatbot's ability to navigate and interpret complex data relationships, offering precise and relevant responses.

2.2 Metadata Generation Pipeline Automation

Following the completion of the schema and relationship definitions, we embarked on the generation of mock data and automated the entire pipeline. Initially, we facilitated interaction between Python and ChatGPT through LangChain and the ChatGPT API, followed by the gathering of publicly available reports online. Due to a lack of existing

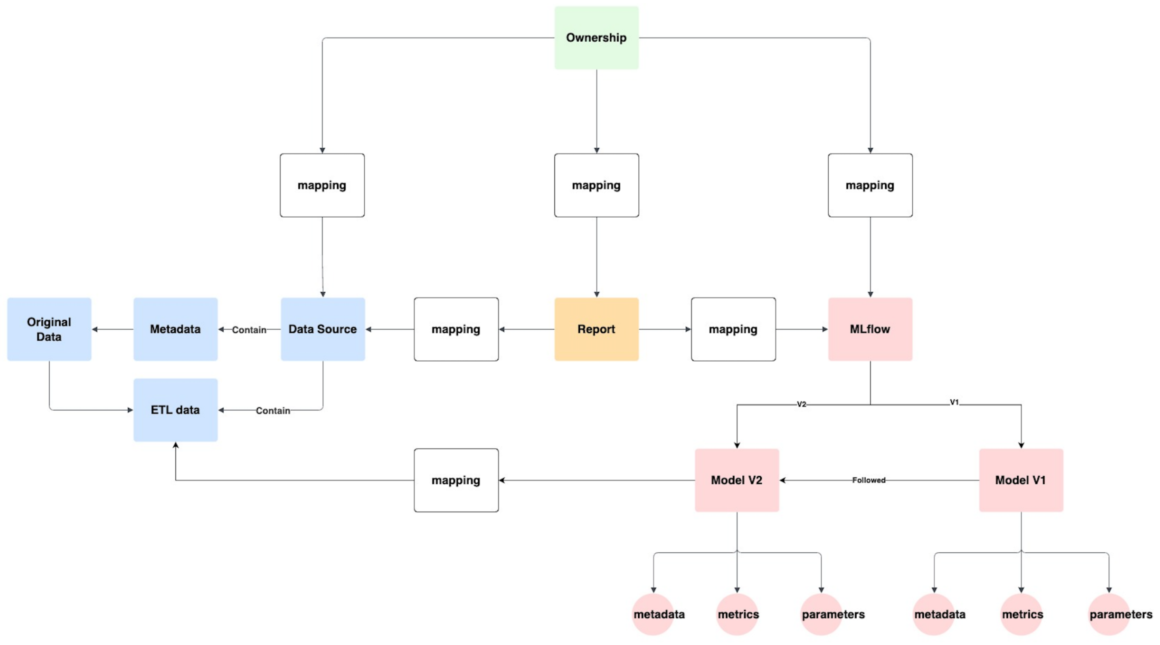


Figure 4: Diagram of A Relationship Network

data regarding MLflow, data sources, ownership, and ETL pipelines, we began by manually creating pertinent and rational data for these reports, along with the corresponding data in JSON and Cypher formats. This crafted data was then input into 2 Python orchestrator for fine-tuning the conversion of unstructured data into JSON files and subsequently transforming JSON files into Cypher files. Additionally, we employed another Python orchestrator to fine-tune the generation of data from the crafted data. After all fine-tuning processes were completed, we engaged in prompt engineering with ChatGPT to produce a series of highly logical and consistent mock data.
Figure 5 illustrates the workflow diagram of the entire pipeline. Throughout the process of generation and automation, our goal was to fine-tune three Python orchestrators endowed with the following distinct functionalities:

1. Transforming unstructured data into JSON files.

2. Converting JSON files into Cypher files.

3. Generating reports and associated logically consistent data related to MLflow, data sources, etc.

The fourth stage in our process represents our automated pipeline. By implementing such a workflow, we are able to efficiently produce a large volume of high-quality data. This foundation is crucial for the subsequent creation of a graph database and the fine-tuning of the Q&A chatbot, setting a solid base for enhanced data management and interaction capabilities.

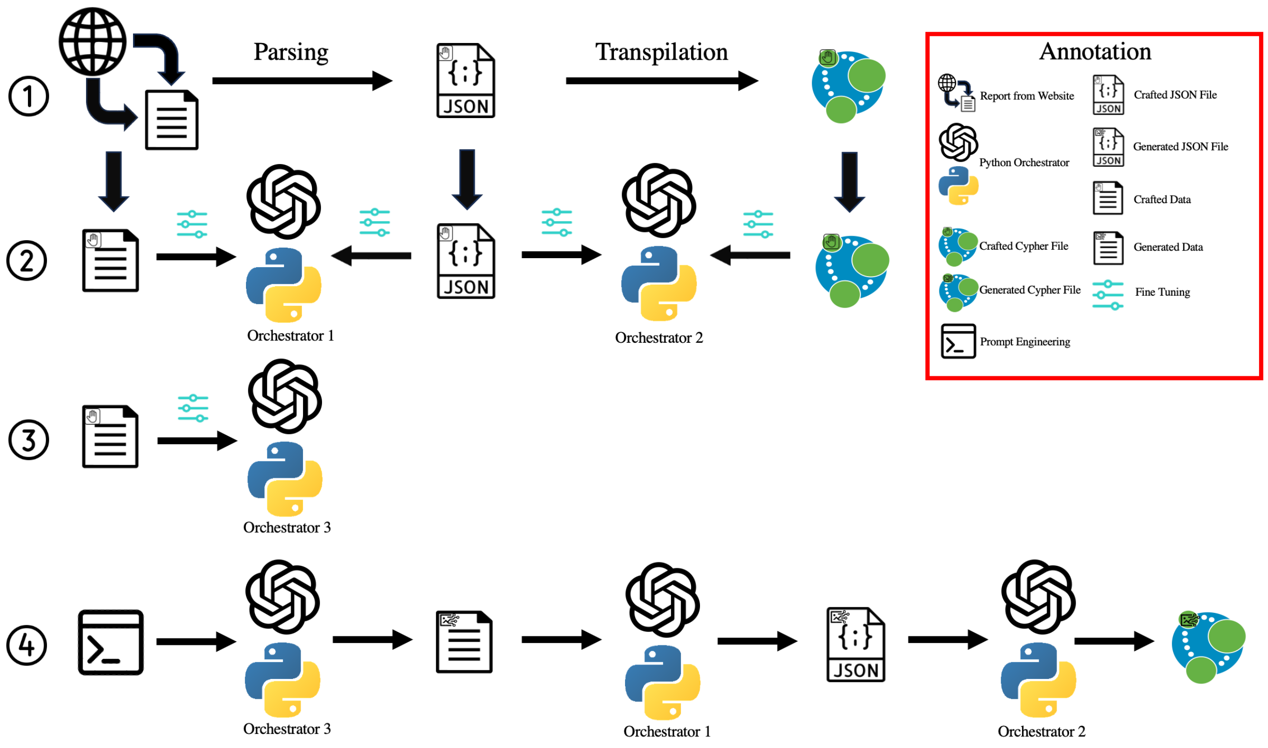


Figure 5: Flowchart of Mock Model Data Generation and Pipeline Automation

3 Graph Database Construction

3.1 Neo4j Local Database Construction

Upon the successful generation of mock metadata, we required a database capable of storing this information. Specifically, if we consider these entities as nodes and their relationships as edges, then these data and their connections form a graph. Consequently, we opted to use a graph database for storage. In the initial phase of our project, we employed Neo4j Desktop, which does not limit the number of nodes and relationships created, thereby facilitating a more efficient exploration of mock metadata storage methods.

3.2 Metadata Importation

We imported the data into Neo4j Desktop thourgh generated Cypher files, and Figure 6 displays the graph structure of one set of the data. In the diagram, the green, red, and yellow nodes represent reports, databases, and ML models, respectively, each extending into their corresponding substructures. These three segments are interconnected through purple nodes, which represent mappings, thereby achieving efficient storage of the data and its relationships.

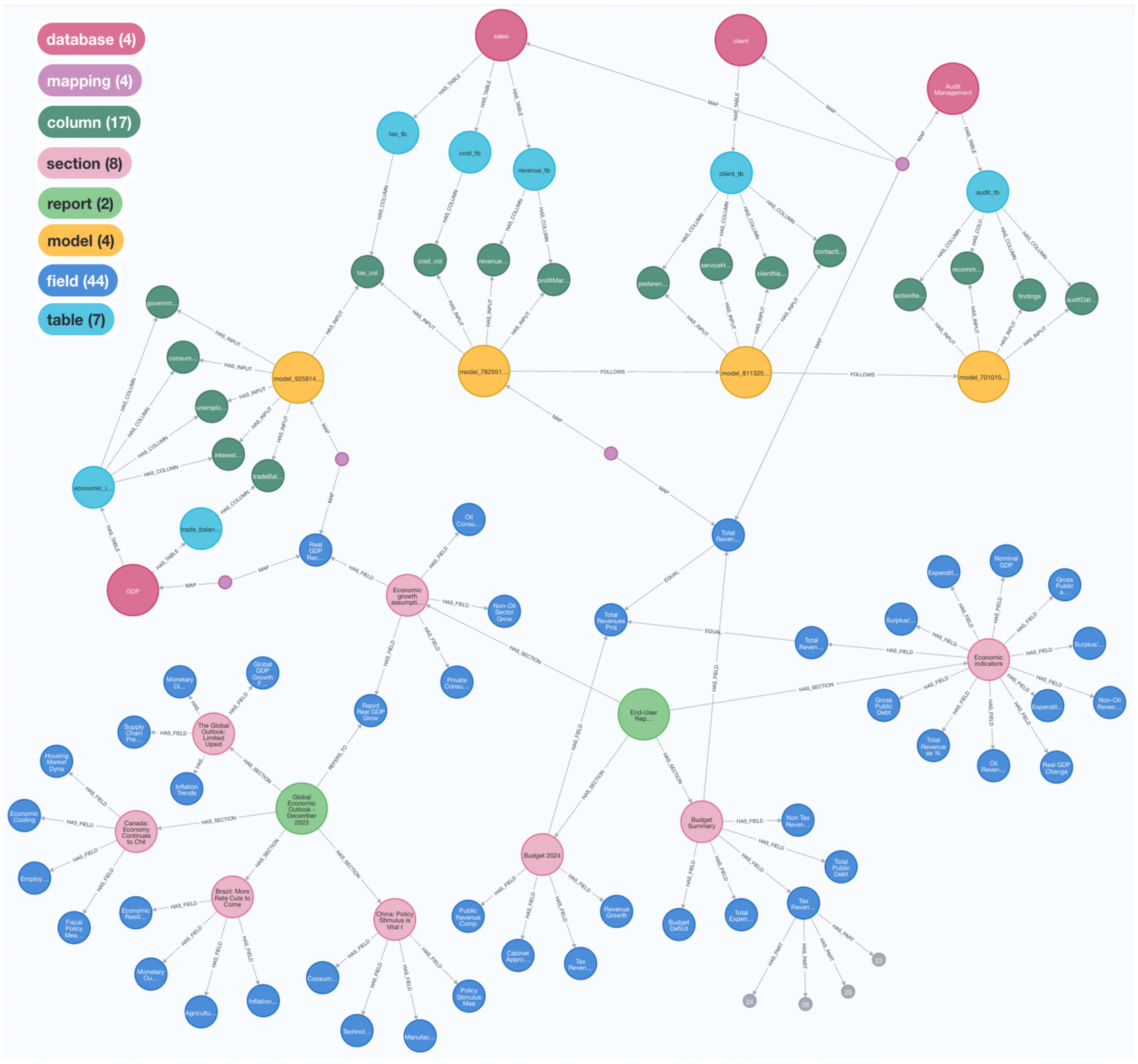


Figure 6: Example of Mock Metadata Imported into Neo4j Graph Database

4 GenAI Powered Chatbot Pre-development

4.1 API setup

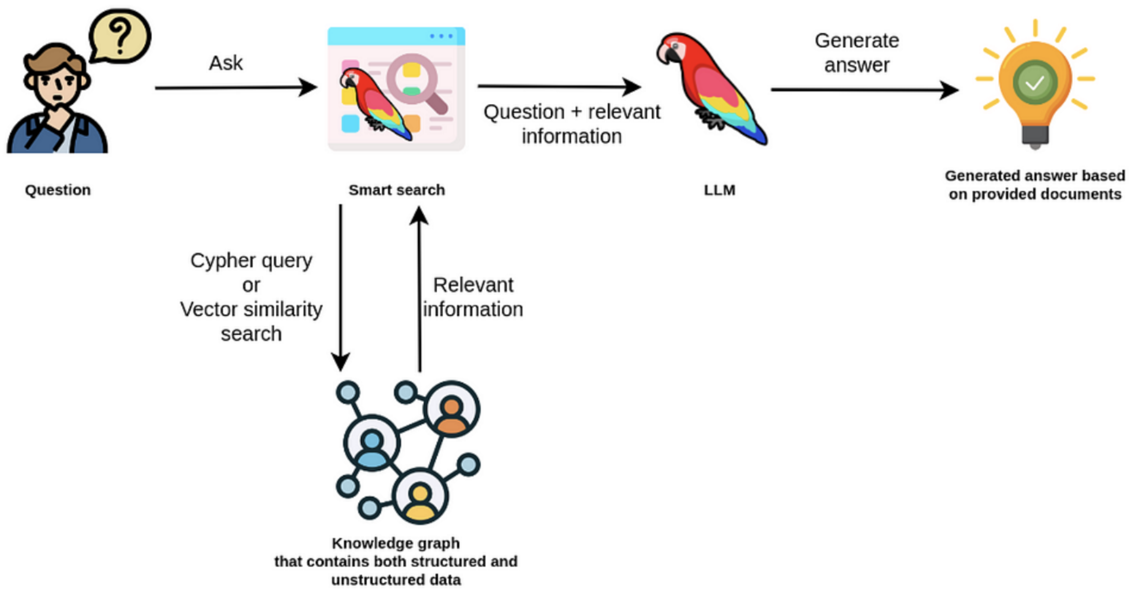
This project requires establishing connections to ChatGPT and the LangChain API. This API setup will serve as the foundation for the chatbot.

4.2 Connection to Graph Database

Combining LangChain with Neo4j utilizes the robust capabilities of Neo4j's graph database to amplify Large Language Models (LLMs) with sophisticated knowledge organization and advanced query functions.

A key method employed was CypherQAChain, a component of LangChain. The process it facilitates is illustrated in Figure 7. Primarily, it converts questions into Cypher queries, runs these queries against the connected Neo4j database, and then uses another LLM to transform the retrieved data back into a natural language response [1].

Following the establishment of the API setup and the connection to the local Neo4j database, we are now able to accurately pull specific statistics from the mock database in response to queries such as "what is the data from the section of the report". Furthermore, through the use of CypherQAChain, these statistics can be seamlessly integrated into sentences that provide clear answers to user questions. These outputs will be shown in the next Use Cases section.



Ask
Question
Smart search
Question + relevant information
LLM
Generate answer
Generated answer based on provided documents
Cypher query or Vector similarity search
Relevant information
Knowledge graph that contains both structured and unstructured data

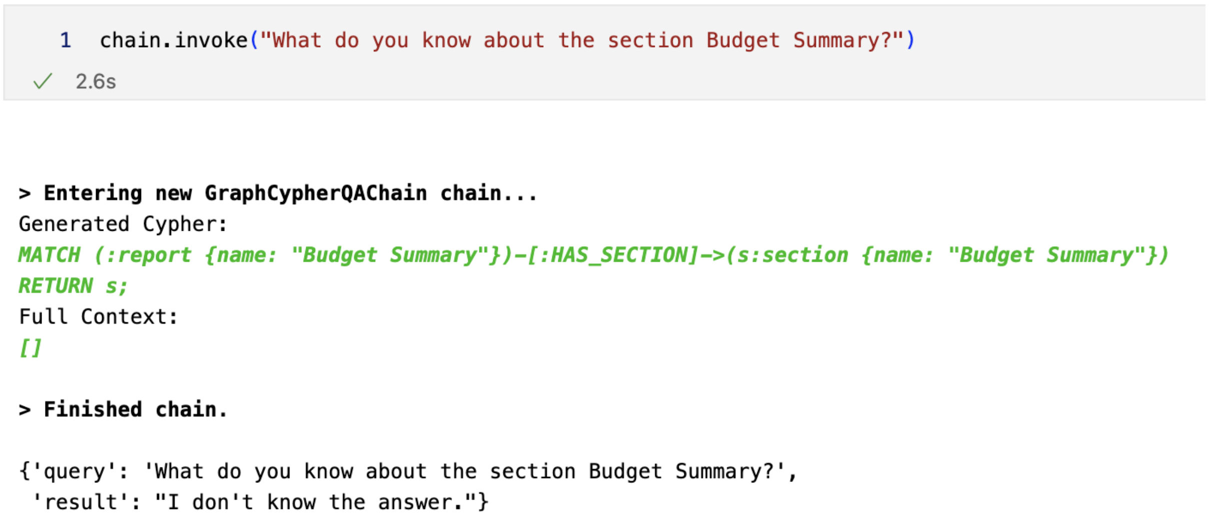
Figure 7: Langchain with Connection to Neo4j Database [1]

5 Use Cases

5.1 Information Extraction

The fundamental use case of our chatbot is designed to facilitate users in swiftly accessing essential information directly from the graph database. This functionality significantly enhances operational efficiency by enabling the chatbot to perform information retrievals. These are straightforward queries that pull specific data points from the graph database, bypassing the need for intricate interpretation or complex reasoning processes. For example, users can effortlessly retrieve basic information about reports, such as their names, associated business groups, maintainers, and entitlements. Similarly, for models, the chatbot can provide details like names, versions, input and output columns, creation dates, and authors. This direct access to critical data simplifies information retrieval, making it an invaluable tool for users who require quick and precise data without navigating through complex database structures.

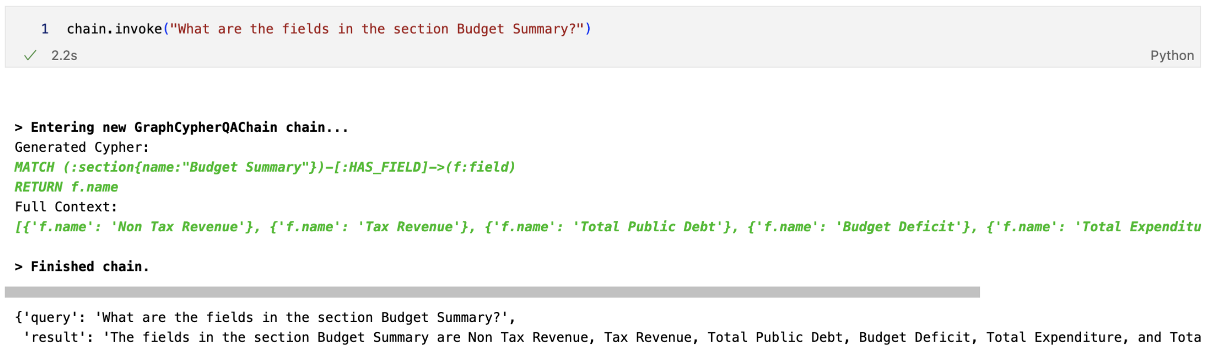
This step can be accomplished by simply connecting the graph database with the API. However, the response from this base case model can be very unstable. For example, when we pose the query "What do you know about the section Budget Summary?" as shown in Figure 8, the base case model does not provide an answer. This indicates that GPT was unable to generate a Cypher query to extract the information needed to respond to the prompt. However, in some cases, the base case model can successfully generate an answer. For instance, when we ask the query "What are the fields in the section Budget Summary?" as shown in Figure 9, the base case model provides the correct answer. We suspect that the way questions are phrased prompts GPT to generate the correct response. Therefore, one limitation of the base case model lies in the formulation of the prompt.



```
chain.invoke("What do you know about the section Budget Summary?")
✓ 2.6s
```

```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (:report {name: "Budget Summary"})-[:HAS_SECTION]->(s:section {name: "Budget Summary"})
RETURN s;
Full Context:
[]

> Finished chain.

{'query': 'What do you know about the section Budget Summary?',
 'result': "I don't know the answer."}
```

Figure 8: Base Case Model: Bad Response



```
chain.invoke("What are the fields in the section Budget Summary?")
✓ 2.2s                                                                    Python
```

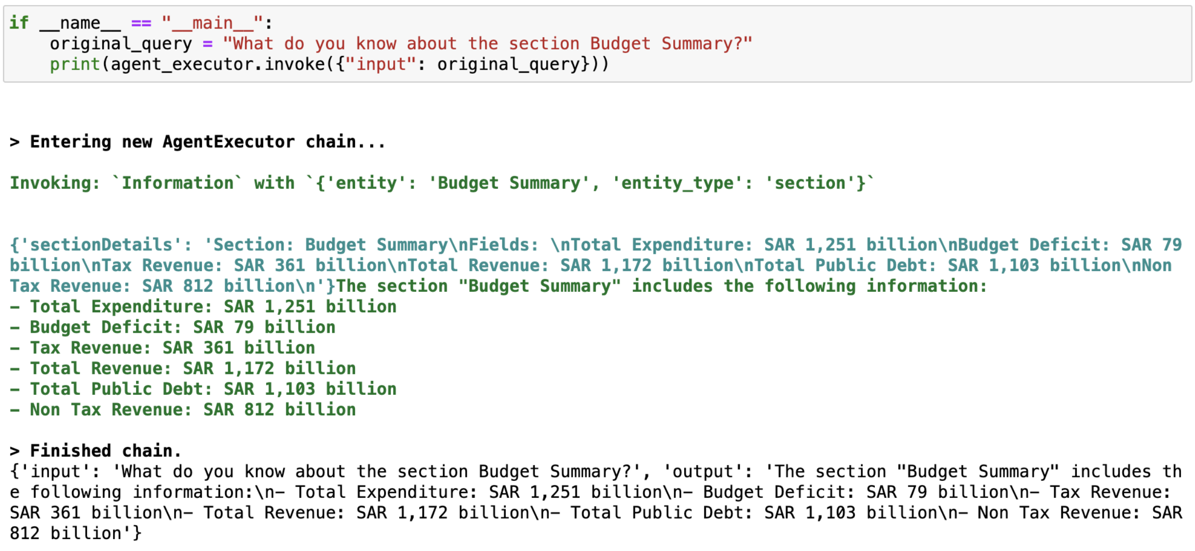
```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (:section{name:"Budget Summary"})-[:HAS_FIELD]->(f:field)
RETURN f.name
Full Context:
[{'f.name': 'Non Tax Revenue'}, {'f.name': 'Tax Revenue'}, {'f.name': 'Total Public Debt'}, {'f.name': 'Budget Deficit'}, {'f.name': 'Total Expenditu

> Finished chain.

{'query': 'What are the fields in the section Budget Summary?',
 'result': 'The fields in the section Budget Summary are Non Tax Revenue, Tax Revenue, Total Public Debt, Budget Deficit, Total Expenditure, and Tota
```

Figure 9: Base Case Model: Good Response

5.2 Classification Layer and Back-and-Forth Interaction

When a user asks a question, the chatbot uses a system to understand the question's type. This classification layer utilizes a combination of natural language processing (NLP) techniques and machine learning models to understand the intent behind a user's question. Once a query is classified, the chatbot identifies the appropriate path to retrieve the answer from the corresponding section of the graph database, ensuring accuracy and relevance in the information provided.

Inspired by the article ‘Enhancing Interaction between Language Models and Graph Databases via a Semantic Layer’ [2], we developed an information tool by hand. This tool guarantees that users receive the most current and pertinent information during the retrieval process. As illustrated in Figure 10, thanks to the classification layer, the LLM can deliver accurate and consistent responses to user inquiries, unlike the scenario depicted in Figure 8.



```
if __name__ == "__main__":
    original_query = "What do you know about the section Budget Summary?"
    print(agent_executor.invoke({"input": original_query}))
```

```
> Entering new AgentExecutor chain...

Invoking: `Information` with `{'entity': 'Budget Summary', 'entity_type': 'section'}`


{'sectionDetails': 'Section: Budget Summary\nFields: \nTotal Expenditure: SAR 1,251 billion\nBudget Deficit: SAR 79
billion\nTax Revenue: SAR 361 billion\nTotal Revenue: SAR 1,172 billion\nTotal Public Debt: SAR 1,103 billion\nNon
Tax Revenue: SAR 812 billion\n'}The section "Budget Summary" includes the following information:
- Total Expenditure: SAR 1,251 billion
- Budget Deficit: SAR 79 billion
- Tax Revenue: SAR 361 billion
- Total Revenue: SAR 1,172 billion
- Total Public Debt: SAR 1,103 billion
- Non Tax Revenue: SAR 812 billion

> Finished chain.
{'input': 'What do you know about the section Budget Summary?', 'output': 'The section "Budget Summary" includes th
e following information:\n- Total Expenditure: SAR 1,251 billion\n- Budget Deficit: SAR 79 billion\n- Tax Revenue:
SAR 361 billion\n- Total Revenue: SAR 1,172 billion\n- Total Public Debt: SAR 1,103 billion\n- Non Tax Revenue: SAR
812 billion'}
```

Figure 10: Base Case Model: Good Response with classification layer

Meanwhile, a distinguishing feature of our chatbot is its capability for back-and-forth interaction. When a query cannot be satisfactorily addressed due to the limitations within the graph database or if the question falls outside the predefined categories, the chatbot will engage in a dialogue with the user. This interaction involves informing the user that the requested information could not be found and encouraging them to refine their query or ask additional questions. This iterative process not only enhances the user experience, but also provides valuable insights to further improve the performance of the chatbot and the comprehensiveness of the graph database.

The Classification Layer is designed with adaptability in mind, learning from each interaction to refine its categorization algorithms and enhance its understanding of user queries. This continuous learning process ensures that the chatbot remains responsive to the evolving needs of users, improving its accuracy and efficiency over time. In short, our chatbot is smart enough to understand your questions, look for answers in a complex database, and keep talking to you if it gets stuck. It’s always learning, making sure it can help you better each time you use it.

5.2.1 Classification Cases Based on User Types

Different users may ask different questions. In the context of our project, it's important to consider the diverse needs and objectives of various users interacting with the chatbot. Each user, depending on their role within the organization, may seek specific types of information or require different levels of detail from the graph database.

A data analyst's primary concern often revolves around ensuring that the data feeding into models complies with established data governance policies. This necessity becomes crucial when models are used for critical decision-making or when data sensitivity is high. Our chatbot streamlines this verification process for the Data Analyst. When tasked with assessing the data sources for a particular model, the Analyst can query the chatbot to list these sources.

On the other hand, a machine learning engineer may be interested in understanding what other models or reports depend on Model A to assess the potential impact of the changes before making changes to it. The chatbot can access the graph database to find all nodes (models and reports) that have dependencies on Model A. It provides a summarized list of these dependent entities, along with a brief description of the nature of their dependency, helping the data scientist gauge the impact of potential changes.

Data engineers often needs to understand how modifications to databases, tables, or columns might impact dependent systems, including machine learning models. Before implementing any changes to the data structure, a data engineer requires a comprehensive understanding of which specific databases, tables, and columns are utilized by each model, and the chatbot can significantly streamline this process for data engineers.

Business analysts often engage in dissecting the lineage of reports to understand the origins of data, the processes involved in shaping the final output, and how changes in data sources or processing steps might influence the accuracy and reliability of reports. This understanding is crucial for ensuring the integrity of data-driven decision-making processes within an organization. The chatbot provides an invaluable tool for business analysts by offering immediate access to detailed report lineage information.

6 Future Steps

6.1 Measuring Performance

In the development of our Retrieval Augmented Generation (RAG)-based GPT chatbot, challenges that could impact the model's performance are considered, such as the accuracy of information retrieval, the quality of the response, and the effective management of erroneous data generation (commonly referred to as hallucinations). The key matrix under consideration is response accuracy. Given the lack of standardized performance metrics directly applicable to our unique context, our approach towards evaluating response accuracy involves a straightforward method. We've developed a strategy where we create questions that already have answers set up in our graph database. This strategy enables a direct assessment of the chatbot's capability to autonomously generate Cypher queries and accurately extract the requisite information.

To quantify the enhancement in performance from the baseline to the more advanced model, we propose calculating the accuracy rate. This rate represents the proportion of instances wherein the chatbot successfully provides correct answers, as pre-defined in the graph database. By employing this metric, we aim to facilitate a direct and quantitative comparison of the improvements in response accuracy between the baseline and the advanced iterations of our chatbot.

6.2 Chatbot Interface Development via Streamlit

To enhance user interactions and provide a seamless experience, we plan to build a Graphical User Interface (GUI) for our chatbot with Streamlit. The interface is designed to replicate conversational interactions, enabling users to submit their inquiries or commands and directly receive replies from the chatbot. This setup is intended to enhance the accessibility and usability of the chatbot, making it suitable for users with varying levels of technical expertise.

6.3 Tackling GenAI Hallucinations via Prompt Engineering

One significant challenge faced by GenAI chatbots is the occurrence of hallucinations, where the generated content may include inaccurate information, make unfounded assumptions, or produce nonsensical outputs. To address this issue and improve the reliability of GenAI outputs, we propose the strategy of prompt engineering. This approach involves crafting the input text (or ”prompt”) for a GenAI model in a way that directs the model towards generating outputs that are more accurate, relevant, and coherent. By thoughtfully designing prompts - taking into account aspects such as clarity, specificity, and context — we can guide GenAI models to a better understanding of the intended task and minimize the risk of producing hallucinated content.

7 Appendix

Everyone equally contributes to the creation of the graph database as well as the development of the chatbot.

References

[1] Neo4j Labs. (n.d.). *LangChain Neo4j Integration.* Retrieved March 13, 2024, from https://neo4j.com/labs/genai-ecosystem/langchain/

[2] Bratanic, T. (2024, January 19). *Enhancing Interaction between Language Models and Graph Databases via a Semantic Layer.* Medium. https://towardsdatascience.com/enhancing-interaction-between-language-models-and-graph-databases-via-a-semantic-layer-0a78ad3eba49