COLUMBIA UNIVERSITY
DATA SCIENCE INSTITUTE

Model Metadata Chatbot with GenAI

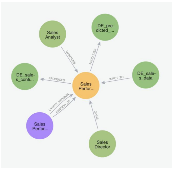
Daniel Clepper, William Holden, Shreya Kurdukar, Selim Shaalan, Keerthi Srilakshmidaran, Akshay Venkatesan
Industry Mentors: Ben Carper, Jim Leach, Sydney Son, Chengwei Wang

Data Science Capstone Project with KPMG

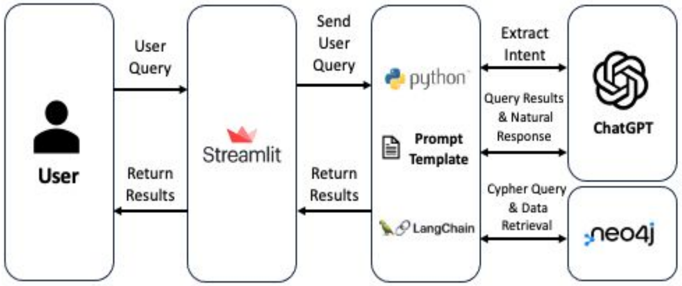
Introduction

Data-driven decision making is the driving force of business in the digital age. However, a reliance on black-box methodologies with little transparency presents a myriad of risks. This project aims to mitigate these risks by providing easy access to digital records. We took a pre-existing metadata chatbot model (created through a collaboration between the Columbia Data Science Institute and KPMG in Spring 2024) and implemented several improvements to enhance the accuracy, relevance, and robustness of chatbot responses.

Our chatbot version employs **GraphRAG** to capture nuanced variable relationships across KPMG databases, models, and reports. Via the chatbot UI, users can ask questions about company data and receive natural language responses, occasionally accompanied by informative relationship graphs.



Chatbot Configuration



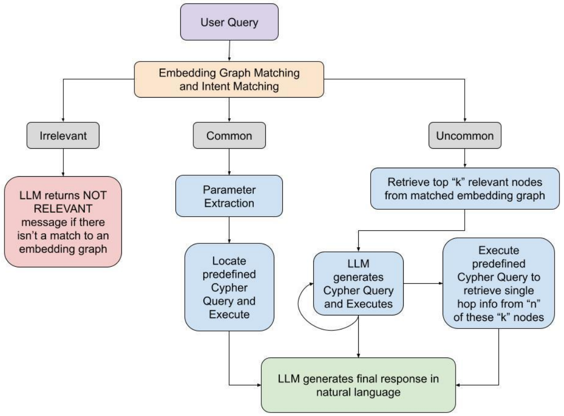
The diagram illustrates the service components of last semester's chatbot and our modifications. The RAG chatbot, built with **Streamlit**, **LangChain's Neo4j integration**, and **OpenAI's GPT-4o**, uses Streamlit as the interface connecting backend and frontend. LangChain integrates with Neo4j to query a graph database using natural language via the **GraphCypherQAChain** module, which translates user queries into Cypher commands.

Our primary modifications include the following:

1. Node Embeddings in the Graph & **RetrievalQA** based retrieval
2. Retrieved Contexts passed into GraphCypherQAChain
3. **Self Reflection** for Query Improvement
4. Single Hop Context Retrieval
5. Refined Prompt Templates
6. LLM Adjudication for Correctness Evaluation

Workflow

1. **Irrelevant Questions:** User is prompted to ask a more relevant question. If the second question is relevant, the intent matching algorithm categorizes it as *common* or *uncommon*.
2. **Common Questions:** A predefined Cypher query is used to retrieve relevant information from the Neo4j database and an LLM generates a natural language response.
3. **Uncommon Questions:** After matching the user query to its closest embedding graph, top *k* contexts are retrieved. Using these, the query, and database schema, GraphCypherQAChain will try to create and execute a Cypher query. The query will be regenerated and executed up to 5 times if it does not retrieve information necessary to answer the question. If after the 5th attempt, the query is still unsuccessful, a predefined Cypher query is used to retrieve information from the nodes that are a *single hop* away from the n=2 most relevant nodes from the previously obtained context.

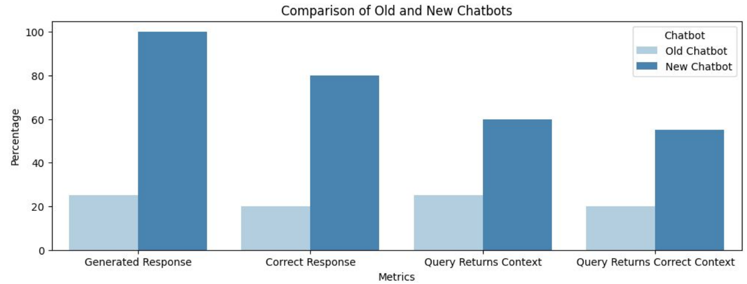


Performance

First, a test set was manually created with 20 (question, query, ground truth answer) triples.

Then, we set up an evaluation pipeline using **G-Eval** based LLM Adjudication to assess correctness across 4 metrics:

- # generated responses
- # correct responses
- # queries returning some context
- # queries returning correct context



The results comparing the old to new model are shown above. As seen, the old chatbot only generated responses for 25% with 20% total correct, while our new chatbot generated responses for 100% of the questions with **80% total correct.** In general, we see that our version of the chatbot was able to generate Cypher queries for 60% of the user questions, compared to the 25% of the old model.

Conclusion

Through our modifications to the chatbot workflow, specifically through employing GraphRAG methodology and improved prompt handling, we expanded the range of question types the system could address and significantly improved its ability to retrieve information from KPMG's data. Particularly, our model is robust on trying different approaches to retrieve contexts, and is quite good at generating matching Cypher queries.

Next Steps

1. Additional prompt engineering to account for exact punctuation/language differences
2. Account for large token issues
3. Implement guardrails
4. Additional file with metadata about the graph
5. Fine tune LLM Adjudicator prompts