SupplyChainInsights: A Comprehensive Report on RAG + Fine Tuning Architecture

Abhishek Shankar - 002751339

August 6, 2024

Contents

1 Introduction

This report presents an in-depth analysis of the SupplyChainInsights project, which focuses on utilizing neo4j graph databases, advanced graph data science techniques, and retrieval-augmented generation (RAG) for enhanced data exploration and insights into global health commodity supply chains. Additionally, the project includes testing out various fine tuned LLMs and integrating it into the application. This document outlines the project's objectives, key features, architecture, and the challenges faced during its development. The source code is available at GitHub
It is very hard for a company to track its supply chain. The central idea of this product is to provide transparency to the entire organisation or company about products. Through a simple chat interface and utilising Generative AI can make access to data very useful.

SupplyChainInsights is a project aimed at analyzing global health commodity supply chains, particularly Antiretroviral (ARV) and HIV lab shipments. By leveraging neo4j graph databases and advanced data science techniques, the project provides detailed insights and data retrieval capabilities. The integration of retrieval-augmented generation (RAG) and fine-tuning of models ensures robust data handling and querying.

At an overview this application allows users to ask natural language questions based on the data - this question depending on the complexity and nature can be presented in simple language either visually or in concise ways.

2 Dataset

The dataset encompasses comprehensive information on ARV and HIV lab shipments, commodity pricing, supply chain expenses, and delivery volumes by country. This rich dataset is pivotal for the project's data analysis and insight generation.

It is important to know this is a sample supply chain dataset, it covers only a small usecase

Access the dataset

3 Key Features

3.1 Neo4j Graph Database

The project constructs a graph database to model and analyze supply chain relationships, enhancing the ability to visualize and query complex data struc-

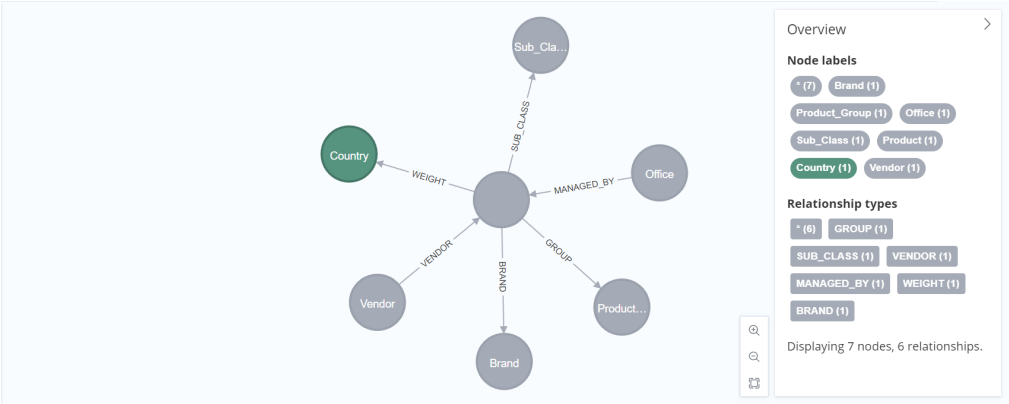


Figure 1: Schema of Neo4j

tures. Neo4j facilitates the structure of data as entities and relationships. (Refer to Figure 1)

3.2 Retrieval-Augmented Generation (RAG)

RAG is implemented for advanced data retrieval and user interaction, enabling the direct extraction of key data aspects using OpenAI embeddings and Cypher QA Chain. This integration allows natural language queries to be converted into Cypher queries, facilitating detailed data access and visualization.

3.3 Fine-Tuned Model

Fine-tuning helps LLMs learn from custom datasets while retaining the important information. This can be useful when data cannot be used to facilitate data privacy. A fine-tuned model using Llama3, Unsloth, and HuggingFace ensures data integrity and efficient querying. The project evaluates several models, including Phi3, Llama3 - 8b, and GPT-3.5, to determine the most effective approach for natural language processing within the application. These models can be used to convert Natural Language to Cypher Queries. These Cypher queries can be used to access the raw data as well as understand the raw data and for internal purposes.

3.4 Performance Monitoring

RAGAS is a framework that helps you evaluate your Retrieval-Augmented Generation (RAG) pipelines. RAG denotes a class of LLM applications that use external data to augment the LLM's context. There are existing tools and frameworks that help you build these pipelines, but evaluating and quantifying your pipeline performance can be challenging. This is where Ragas (RAG Assessment) comes in.

Ragas provides a structured approach to evaluating RAG models by offering a variety of metrics that assess different aspects of the model's performance. By using Ragas, you can gain a detailed understanding of how well your RAG pipeline is performing and identify areas for improvement. The evaluation framework provided here helps in assessing the performance and effectiveness of RAG models in generating accurate and relevant answers by utilizing retrieved contexts.

4 Project Architecture

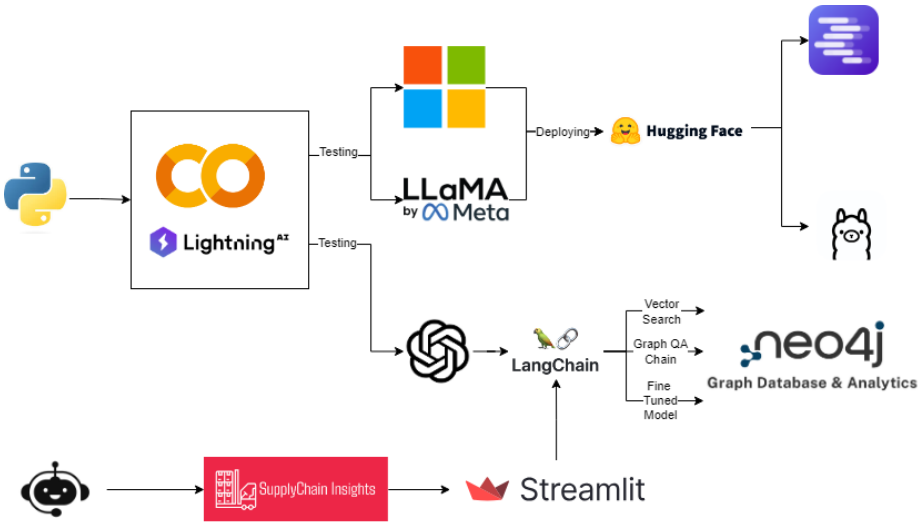


Figure 2: Project Architecture

The architecture integrates various components, including Neo4j, OpenAI, LangChain, and fine-tuned models, to create a cohesive system for data analysis and retrieval.

4.1 Explanation

- The entire application has been developed and tested using Python.
- Utilized Google Colab and Lightening.ai for GPU-based training of Llama and Phi3 models to convert Natural Language to Cypher Queries.
- Cypher queries are used to access raw data and understand how to interact with Neo4j datasets.

- Trained models were deployed on Hugging Face after generating .GGUF files.
- Deployed .GGUF files on Ollama and LMStudio for interaction.
- The fourth model, OpenAI, was fine-tuned using the UI and deployed on the OpenAI platform. This is used in the application due the ease in integrating it with the existing application.
- Utilized LangChain to integrate the OpenAI fine-tuned model, Vector Search, and CypherQAChain, facilitating a multi-agent tool to access each feature.
- Each feature is integrated into a Streamlit application, providing an overall cohesive system for data analysis and retrieval.

5 About RAG

5.1 Vector Search with OpenAI Embeddings

Vector search is a methodology where the natural language questions and the data inside of a database are compared based on their embedding. The data point which is closest to the question is usually the answer. In this case, the product Utilizes OpenAI embeddings for efficient data retrieval, allowing users to query detailed product information based on specific data points. I have created embeddings for important data points which can prove useful to users.

5.2 Cypher QA Chain

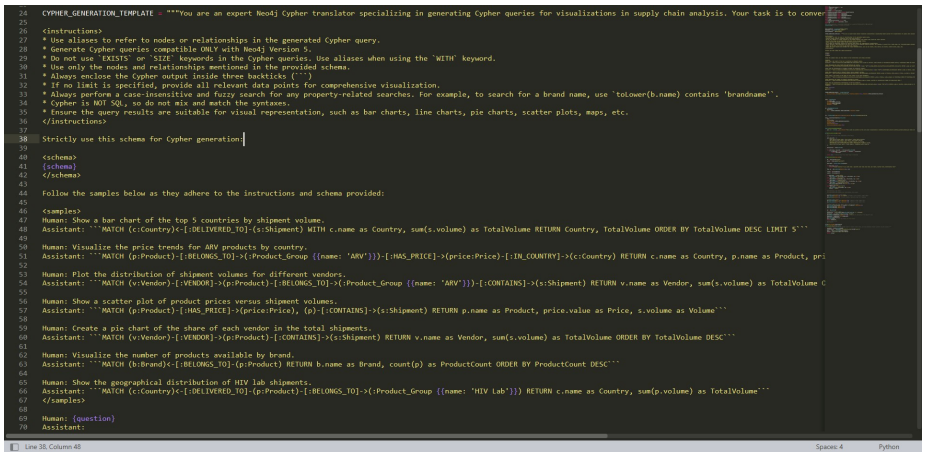
Employs LangChain to convert natural language queries into Cypher queries, enabling comprehensive data access and visualization capabilities. This feature uses Cypher queries to tap into the dataset. Using this I can execute queries and generate answers on the fly. The results obtained from generating the queries can also be used to provide visuals to the user.

An important aspect of RAG is prompt engineering hence in this case I have used a few shot prompting where provide some examples for the LLM to learn from before it starts answering the users questions. it can also be done to steer the model to better performance

6 About Fine Tuning

6.1 Model Evaluation and Fine Tuning

I compared several models by fine-tuning them using Unsloth and processed them on platforms like Lightening.ai and Google Colab. The following models were analyzed:



```
CYPHER_GENERATION_TEMPLATE = """You are an expert Neo4j Cypher translator specializing in generating Cypher queries for visualizations in supply chain analysis. Your task is to conver

<instructions>
* Use aliases to refer to nodes or relationships in the generated Cypher query.
* Generate Cypher queries compatible ONLY with Neo4j Version 5.
* Do not use `EXISTS` or `SIZE` keywords in the Cypher queries. Use aliases when using the `WITH` keyword.
* Use only the nodes and relationships mentioned in the provided schema.
* Always enclose the Cypher output inside three backticks (```)
* If no limit is specified, provide all relevant data points for comprehensive visualization.
* Always perform a case-insensitive and fuzzy search for any property-related searches. For example, to search for a brand name, use `toLower(b.name) contains 'brandname'`.
* Cypher is NOT SQL, so do not mix and match the syntaxes.
* Ensure the query results are suitable for visual representation, such as bar charts, line charts, pie charts, scatter plots, maps, etc.
</instructions>

Strictly use this schema for Cypher generation:

<schema>
{schema}
</schema>

Follow the samples below as they adhere to the instructions and schema provided:

<samples>
Human: Show a bar chart of the top 5 countries by shipment volume.
Assistant: ```MATCH (c:Country)<-[:DELIVERED_TO]-(s:Shipment) WITH c.name as Country, sum(s.volume) as TotalVolume RETURN Country, TotalVolume ORDER BY TotalVolume DESC LIMIT 5```

Human: Visualize the price trends for ARV products by country.
Assistant: ```MATCH (p:Product)-[:BELONGS_TO]->(:Product_Group {{name: 'ARV'}})-[:HAS_PRICE]->(price:Price)-[:IN_COUNTRY]->(c:Country) RETURN c.name as Country, p.name as Product, pri

Human: Plot the distribution of shipment volumes for different vendors.
Assistant: ```MATCH (v:Vendor)-[:VENDOR]->(p:Product)-[:BELONGS_TO]->(:Product_Group {{name: 'ARV'}})-[:CONTAINS]->(s:Shipment) RETURN v.name as Vendor, sum(s.volume) as TotalVolume O

Human: Show a scatter plot of product prices versus shipment volumes.
Assistant: ```MATCH (p:Product)-[:HAS_PRICE]->(price:Price), (p)-[:CONTAINS]->(s:Shipment) RETURN p.name as Product, price.value as Price, s.volume as Volume```

Human: Create a pie chart of the share of each vendor in the total shipments.
Assistant: ```MATCH (v:Vendor)-[:VENDOR]->(p:Product)-[:CONTAINS]->(s:Shipment) RETURN v.name as Vendor, sum(s.volume) as TotalVolume ORDER BY TotalVolume DESC```

Human: Visualize the number of products available by brand.
Assistant: ```MATCH (b:Brand)<-[:BELONGS_TO]-(p:Product) RETURN b.name as Brand, count(p) as ProductCount ORDER BY ProductCount DESC```

Human: Show the geographical distribution of HIV lab shipments.
Assistant: ```MATCH (c:Country)<-[:DELIVERED_TO]-(p:Product)-[:BELONGS_TO]->(:Product_Group {{name: 'HIV Lab'}}) RETURN c.name as Country, sum(p.volume) as TotalVolume```
</samples>

Human: {question}
Assistant:
```

Figure 3: Using Few Shot prompting technique

- Phi3 - SLM from Microsoft
- Llama3 - 8b - Meta's Open Source Model
- Llama3 - 8b by Neo4j Developers
- GPT-3.5

The training and fine-tuning processes are detailed in the following notebooks present in the repository:

- Generating Training Data
- Fine Tuning Phi3
- Fine Tuning Llama3
- Comparing Models Locally

After viewing our models' performance and latency, I observed that while the results were accurate, the time taken to generate the results was long.
It's essential to understand the factors that influence Ollama's performance:

- **Hardware capabilities (CPU, RAM, GPU)**
- **Model size and complexity**
- **Quantization level**
- **Context window size**
- **System configuration and settings**

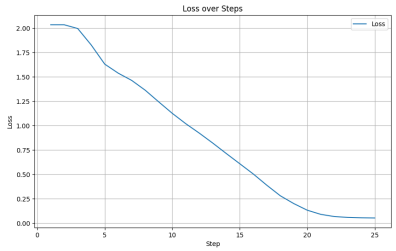


Figure 4: Phi3 Model

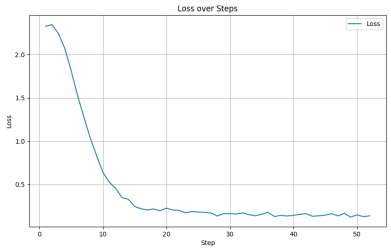


Figure 5: Llamaa3 Model

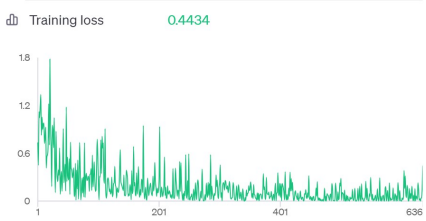


Figure 6: GPT Model

The model size is another important parameter to take into consideration. I used Llama3, which is quite large and could have contributed to the latency. In this case, the configurations of my local environment were as follows - The right GPU was not configured and would have sped up the process.

7 About Performance Monitoring

Performance monitoring of Retrieval-Augmented Generation (RAG) models is crucial for ensuring the effectiveness and efficiency of these systems in generating accurate and relevant answers. In this evaluation, we utilize the Ragas framework, which provides a structured approach to assess various performance metrics of RAG pipelines.

The following metrics are used in our evaluation to gain comprehensive insights into the model's performance:

7.1 Answer Relevancy

Answer relevancy evaluates the relevance of the generated answers to the query, ensuring that the responses are pertinent to the user's question. The metric is calculated using cosine similarity between the embeddings of the generated answer and the original question:

$$\textbf{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^{N} \cos(E_{g_i}, E_o) \tag{1}$$

Where:

- E_{g_i} is the embedding of the generated answer i.
- E_o is the embedding of the original question.
- N is the number of generated answers.

Even though the score will typically range between 0 and 1, it is not mathematically guaranteed due to the nature of cosine similarity ranging from -1 to 1.

7.2 Faithfulness

Faithfulness measures the accuracy of the generated answers in relation to the retrieved contexts. This metric ensures that the answers generated by the LLM are consistent and accurate based on the retrieved information:

$$\textbf{Faithfulness score} = \frac{|\text{Number of claims in the generated answer inferred from given context}|}{|\text{Total number of claims in the generated answer}|} \tag{2}$$

7.3 Latency

Latency measures the response time of the model from input to output, which is crucial for understanding the efficiency of the model in generating answers promptly. Monitoring latency helps in optimizing the performance of RAG models to ensure they meet real-time application requirements.

By leveraging these metrics, we can systematically evaluate the performance of RAG models, identify areas for improvement, and ensure that the models deliver accurate, relevant, and timely responses.

8 Challenges and Solutions

These are the many challenges I faced during the building of this application.

- **Integration of Multiple Technologies**
  - Integrating neo4j, LangChain, and various LLMs required meticulous coordination and testing to ensure seamless functionality.
  - Overcoming compatibility issues and optimizing query execution were major challenges addressed through iterative development and testing.
  - Ensuring the right resources, particularly GPUs, were available was a difficult task. This was solved by using Google Colab and Lightning.ai for training and fine-tuning.
  - Understanding how open-source models function and how to facilitate them in a local system was a significant learning experience.
  - Despite limited computational capacity, I developed a robust framework from scratch to support various LLMs.
- **Fine Tuning Models**
  - Fine-tuning models involved extensive experimentation with different datasets and hyperparameters.
  - Achieving a balance between model accuracy and computational efficiency was a key challenge.
  - Each model's fine-tuning process took a considerable amount of time to ensure all parameters were optimized.
  - Leveraging platforms like Google Colab and Lightning.ai helped streamline the fine-tuning process by providing access to powerful GPUs.
  - Detailed monitoring and adjustment of hyperparameters were necessary to improve model performance and reduce latency.
  - Documenting the fine-tuning process and results provided valuable insights for future model improvements.
- **Resource Constraints**
  - Working with limited computational resources required efficient use of available hardware.
  - GPU configuration played a critical role in enhancing model training speed and performance.
  - Understanding and managing resource limitations was crucial for successful project execution.
  - The following GPU configuration was used to mitigate resource constraints:

9 Conclusion and Future Scope

SupplyChainInsights successfully demonstrates the potential of combining graph databases, RAG, and fine-tuned models for comprehensive data analysis. By leveraging neo4j for graph database capabilities, OpenAI and LangChain for advanced retrieval and natural language processing, and fine-tuning techniques on open-source models, the project provides a robust framework for analyzing global health commodity supply chains. This integration facilitates efficient data retrieval, accurate insights, and improved user interaction.

Future enhancements may include expanding the dataset to cover a broader range of supply chain commodities and scenarios, refining model accuracy through continuous fine-tuning and hyperparameter optimization, and exploring additional use cases such as predictive analytics and real-time monitoring in supply chain management. Further, integrating more sophisticated visualization tools and enhancing the system's scalability and performance will be pivotal in ensuring that SupplyChainInsights remains a cutting-edge tool for supply chain analysis and management.

The project's success underscores the importance of combining multiple advanced technologies to address complex data challenges and paves the way for more innovative applications in the field of supply chain analytics.