ANITA B.ORG 2024

GRACE HOPPER CELEBRATION

me +we

Reeti Sarup

Senior Software Engineer | Intuit

Rishu Kaur

Group Engineering Manager | Intuit

AI Chatbot Masterclass

Building LLM-Powered RAG Chatbots with LangChain & Knowledge Graph

ANITA B.ORG 2024
GRACE HOPPER CELEBRATION

ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Workshop Outcomes

- Practical hands-on experience building a working **RAG-powered AI chatbot**, powered by a locally deployed language model

- Understanding of fundamental concepts of **GenAI**, Retrieval Augmented Generation (**RAG**), Large Language Models (**LLMs**), vector **embeddings**, knowledge **graphs**, **LangChain** and **prompt** engineering

ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

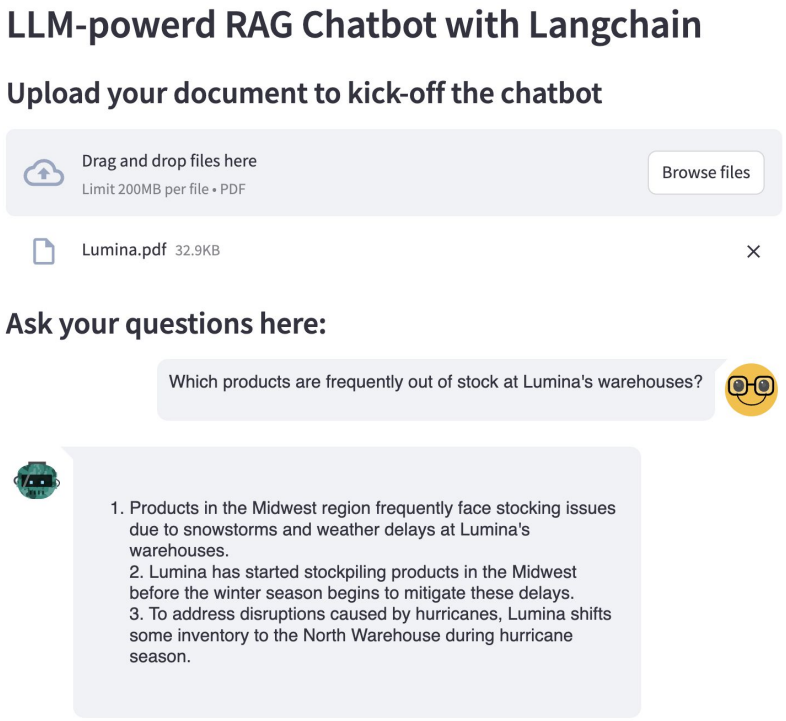
Workshop Content and Prerequisites

Github Link: https://tinyurl.com/ghc2024-aichatbot

Exercise 1

RAG chatbot using LangChain and Vector database

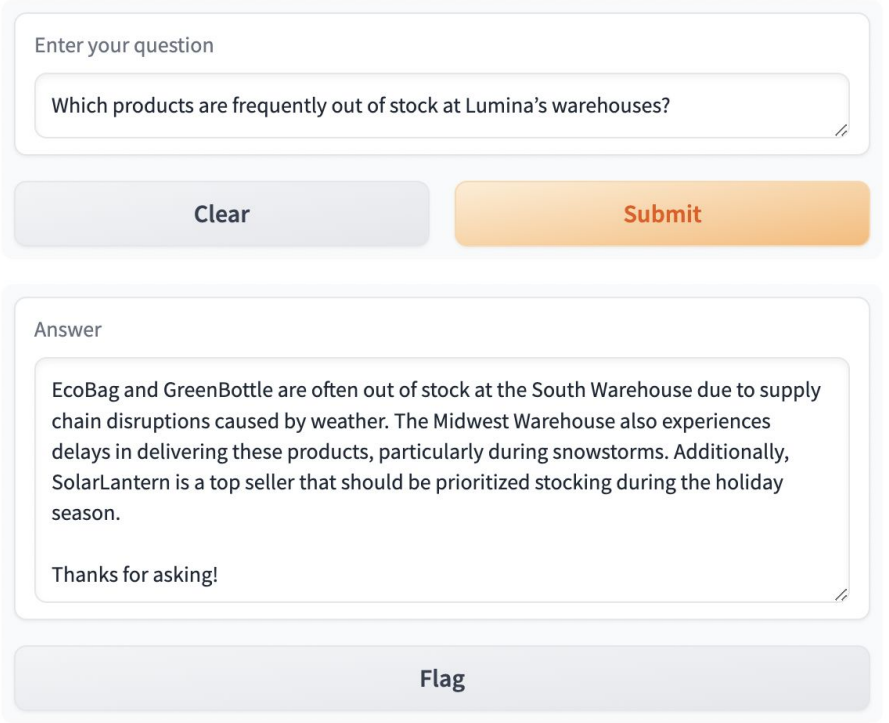
Exercise 1 README



Exercise 2

RAG chatbot using LangChain and Knowledge graph

Exercise 2 README



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Workshop Content and Prerequisites

Github Link: https://tinyurl.com/ghc2024-aichatbot

Pre-requisites Workspace setup starts here

Steps:

Step 1: Install python3 (and pip3)

python version: 3.12.5

https://www.python.org/downloads/macos/

- Download and use the installer
- pip3 should get auto-installed with python.

Step 2: Clone the repository

```
git clone https://github.com/RishuK/grace-hopper-2024-ai-chatbot.git
```

Step 3: Go to each excercise folder and execute the pre-requisites under each folder below

- excercise-1-chatbot-rag-pdf
- excercise-2-knowledge-graph-rag

Option 1:

Setup your local workspace (recommended)

Option 2:
Use a locally running Jupyter notebook and import jupyter files

ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Workshop Content and Prerequisites

Github Link: https://tinyurl.com/ghc2024-aichatbot

Exercise 1 - README

Pre-requisites start here

Pre-requisite 1: Install the Dependencies:

- langchain
- langchain_community
- streamlit - used for building POCs/prototypes About Streamlit
- streamlit-chat Streamlit-chat Installation
- pypdf
- chromadb
- fastembed

```
pip3 install langchain langchain_community streamlit streamlit_chat chromadb pypdf fastembed
```

Pre-requisite 2:Set up Ollama

We need a LLM server which we can easily setup locally and do not have to worry about API keys!

To do this, download Ollama from https://ollama.com/ Ollama can be setup locally and provides multiple models: https://ollama.com/library

We can run a compact model - Mistral-7B

```
ollama pull mistral
```

Pre-requisites end here

Exercise 2 README

Pre-requisite 1: Install the Dependencies:

- langchain
- langchain_community
- langchain_experimental
- langchain_ollama
- langchain_core
- gradio About Gradio

```
pip3 install langchain langchain_community langchain_experimental langchain_ollama langchain_core gradio
```

Pre-requisite 2: Run Llama3.1 via Ollama

We can run a compact model - Llama3.1 8b

```
ollama pull llama3.1:8b
```

Pre-requisite 3: Neo4j Desktop Setup

The step-by-step process for installing and setting up Neo4j Desktop for this exercise are listed below. You can also watch this Neo4j Installation Guide Video for more reference: Neo4j Installation Guide Video

STEP-1: Installation and Local Setup of Neo4j Desktop Setup

1.1. Download the latest free version of Neo4j Desktop from here: https://neo4j.com/download/.

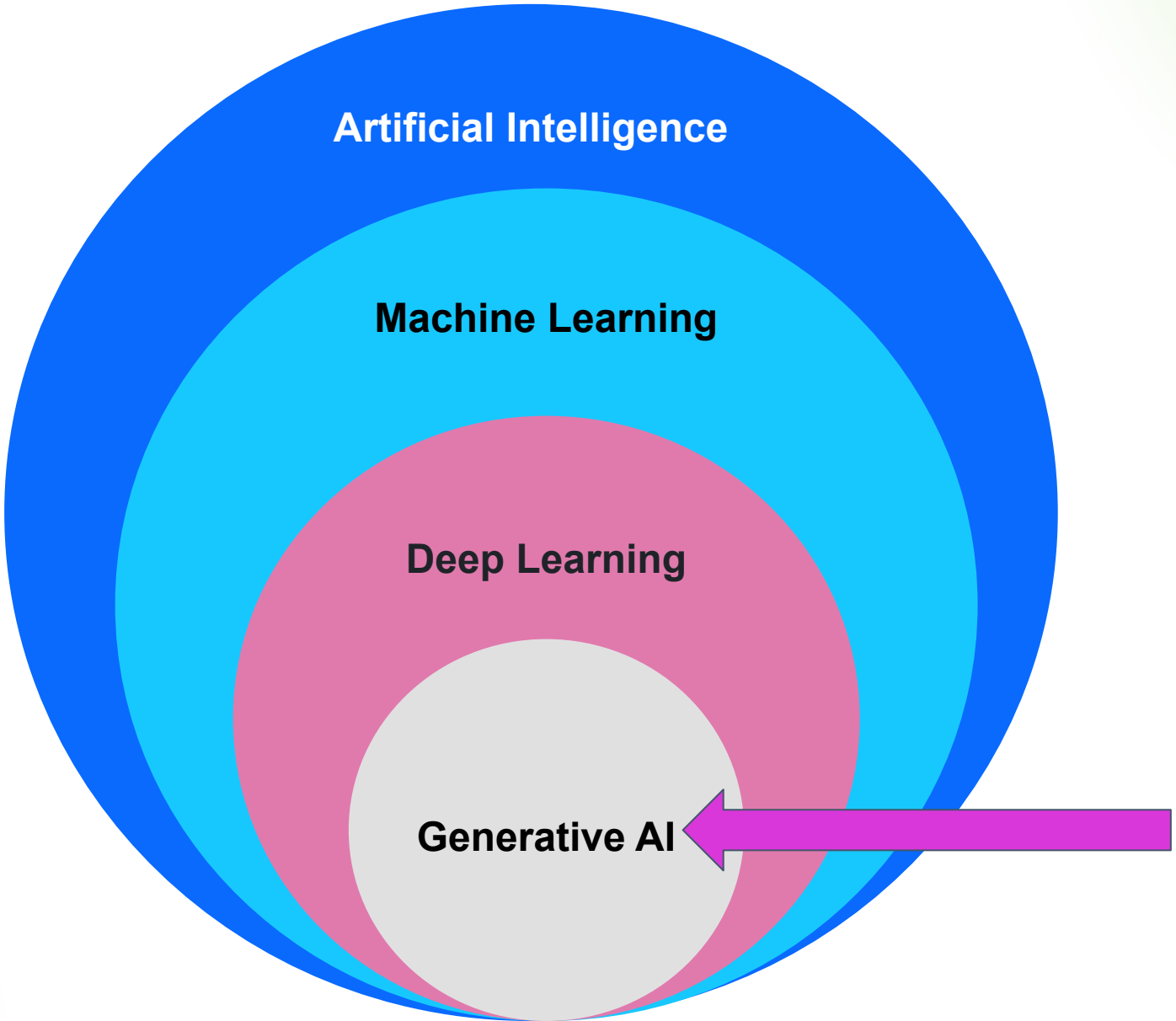
Fill the required form with basic details on the website to start the download.

ANITA B.ORG 2024
GRACE HOPPER CELEBRATION

Core Concepts

me+we

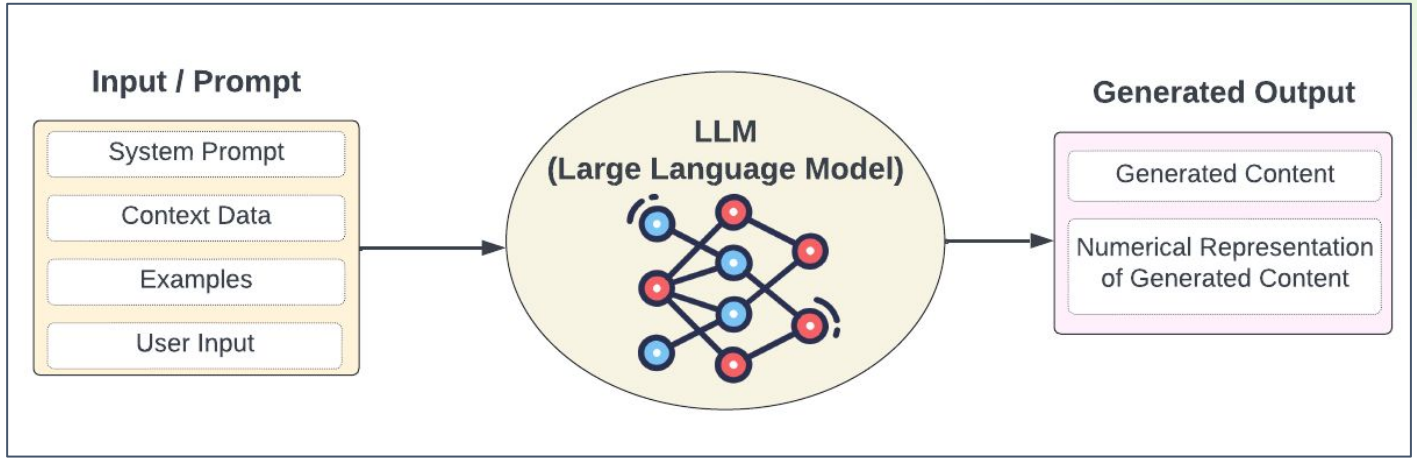
What is GenAI



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

What are Large Language Models (LLMs)

- LLMs are large AI systems based on **deep learning architectures** that are capable of understanding, processing and performing tasks in natural language.
- They are **trained on vast amounts of data** which enables them to understand the patterns of human language.
- LLMs can **save time & effort** by being applied over a variety of generative AI tasks, such as language translation, text summarization, content generation & more.



OpenAI ChatGPT 4.0
GPT by OpenAI

LLama 3
LLaMA by Meta AI

Gemini
Gemini by Google

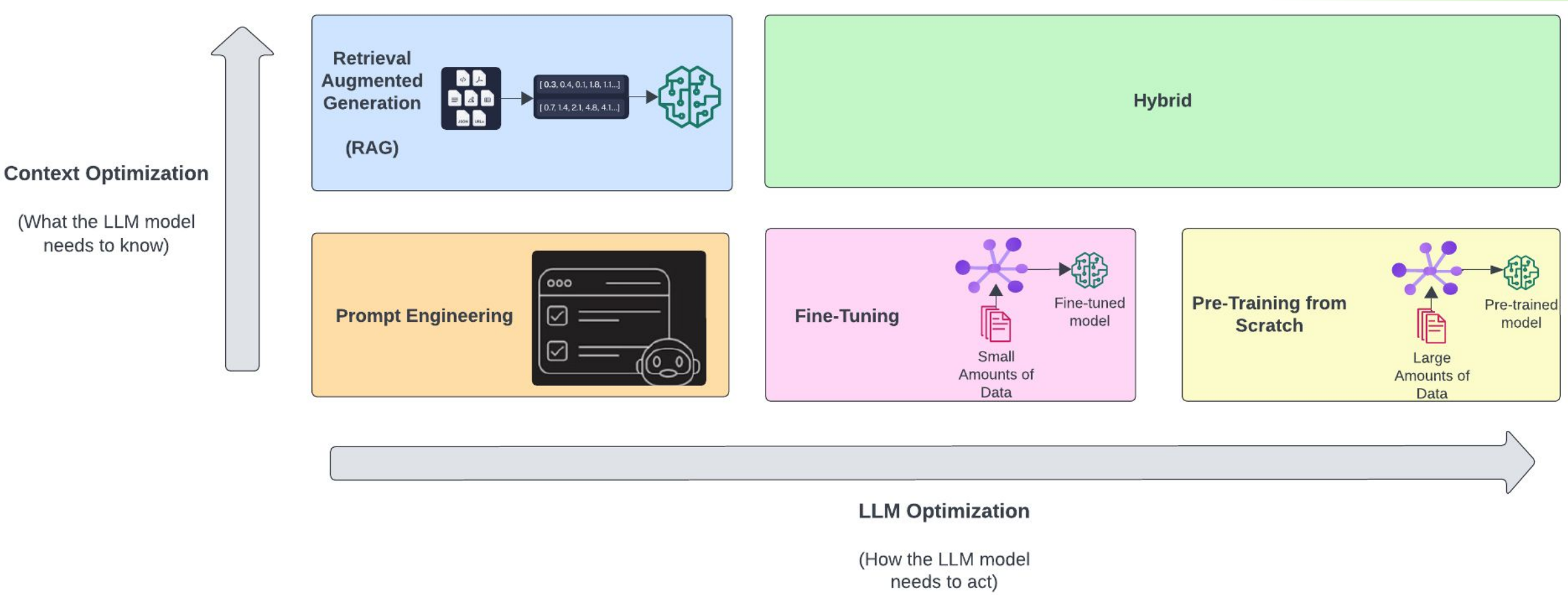
Claude
Claude by Anthropic

MISTRAL AI_
Mistral by MistralAI

DBRX
DBRX by Databricks

ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Overcoming Limitations of LLMs

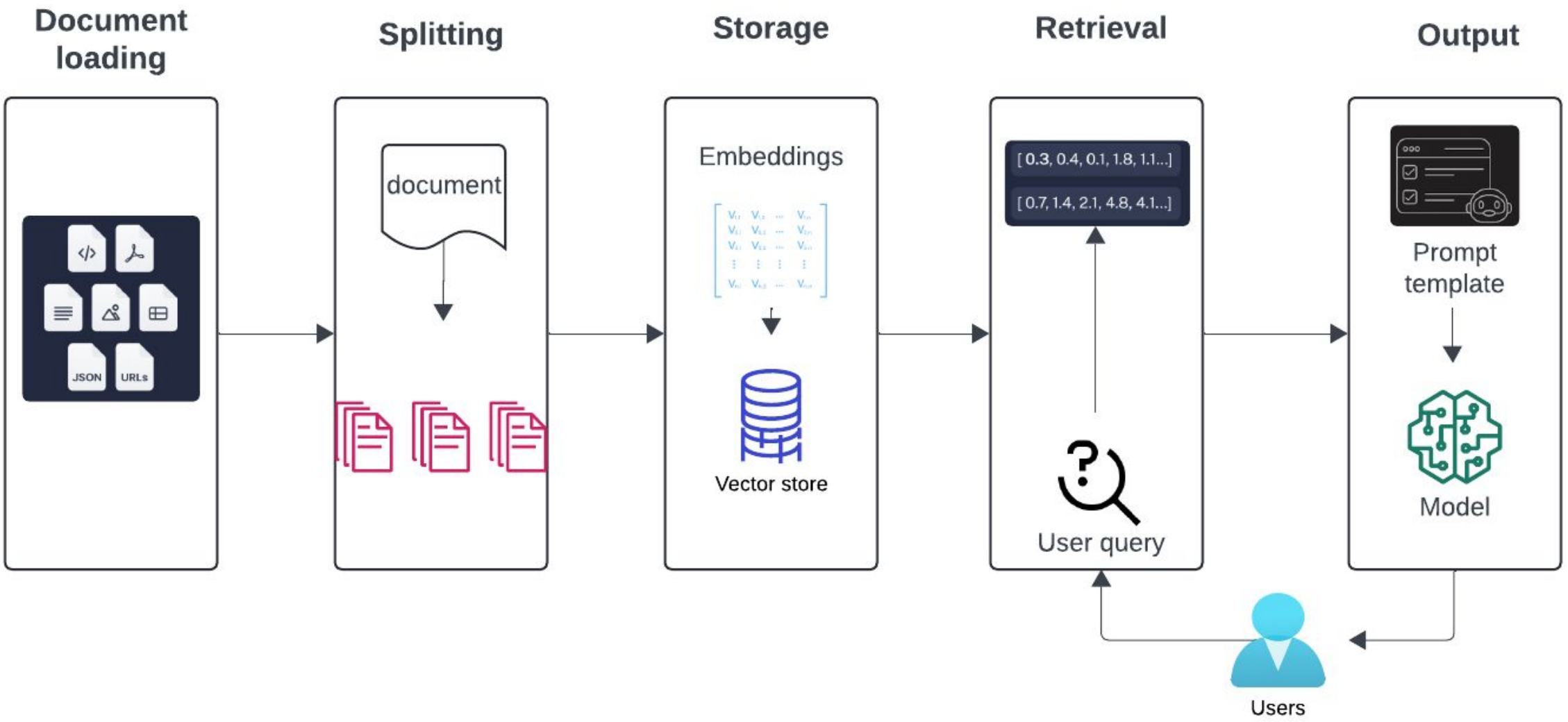


ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Defining Retrieval Augmented Generation (RAG)

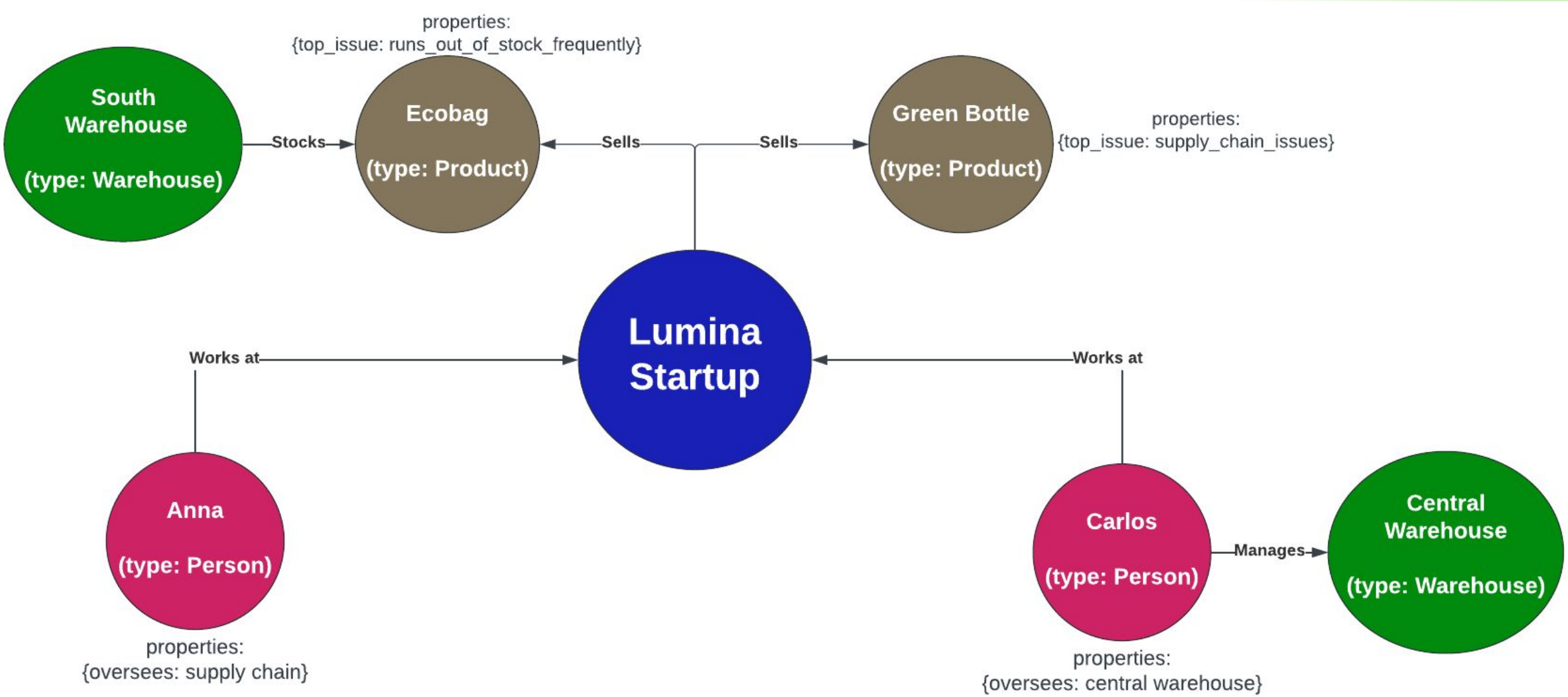
ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

RAG in Action!



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Enriching RAG using Knowledge Graph



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Development Tech Stack

me+we

ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Development Tech Stack Used

Python

Langchain

Ollama

Streamlit

Neo4j

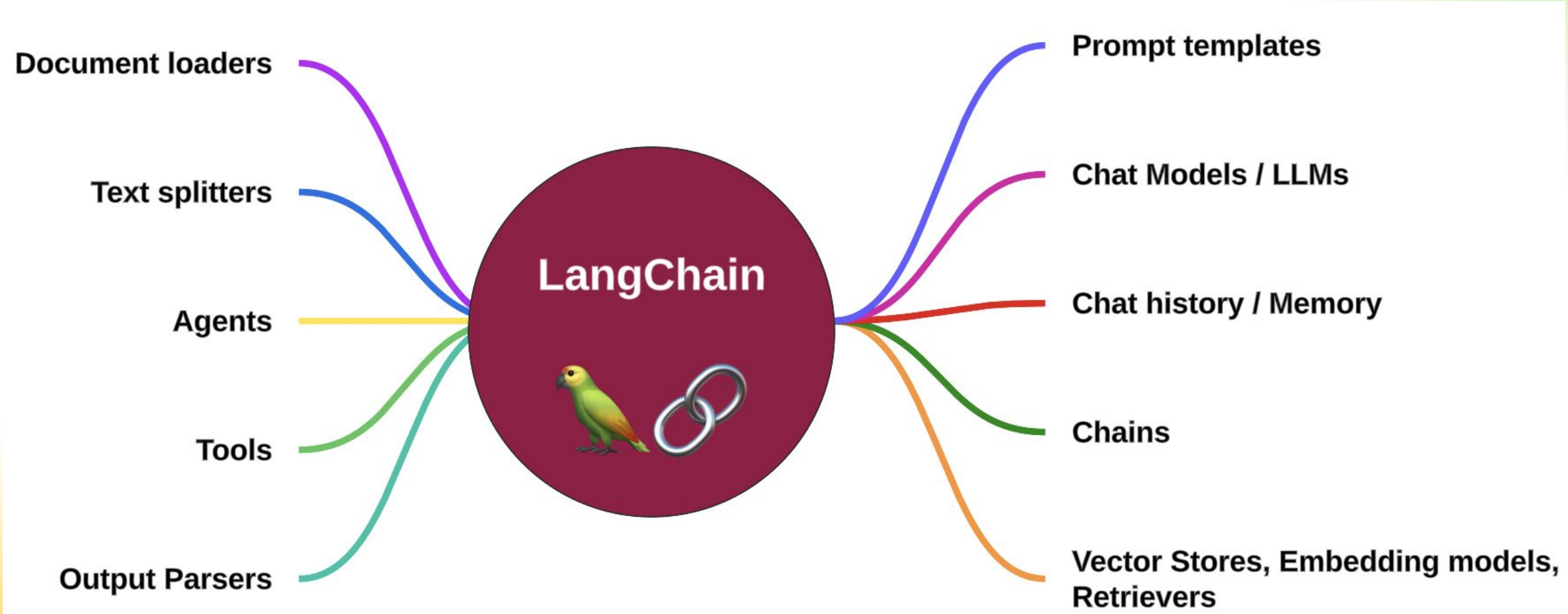
FastEmbed

ChromaDB

Gradio

ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Components of LangChain



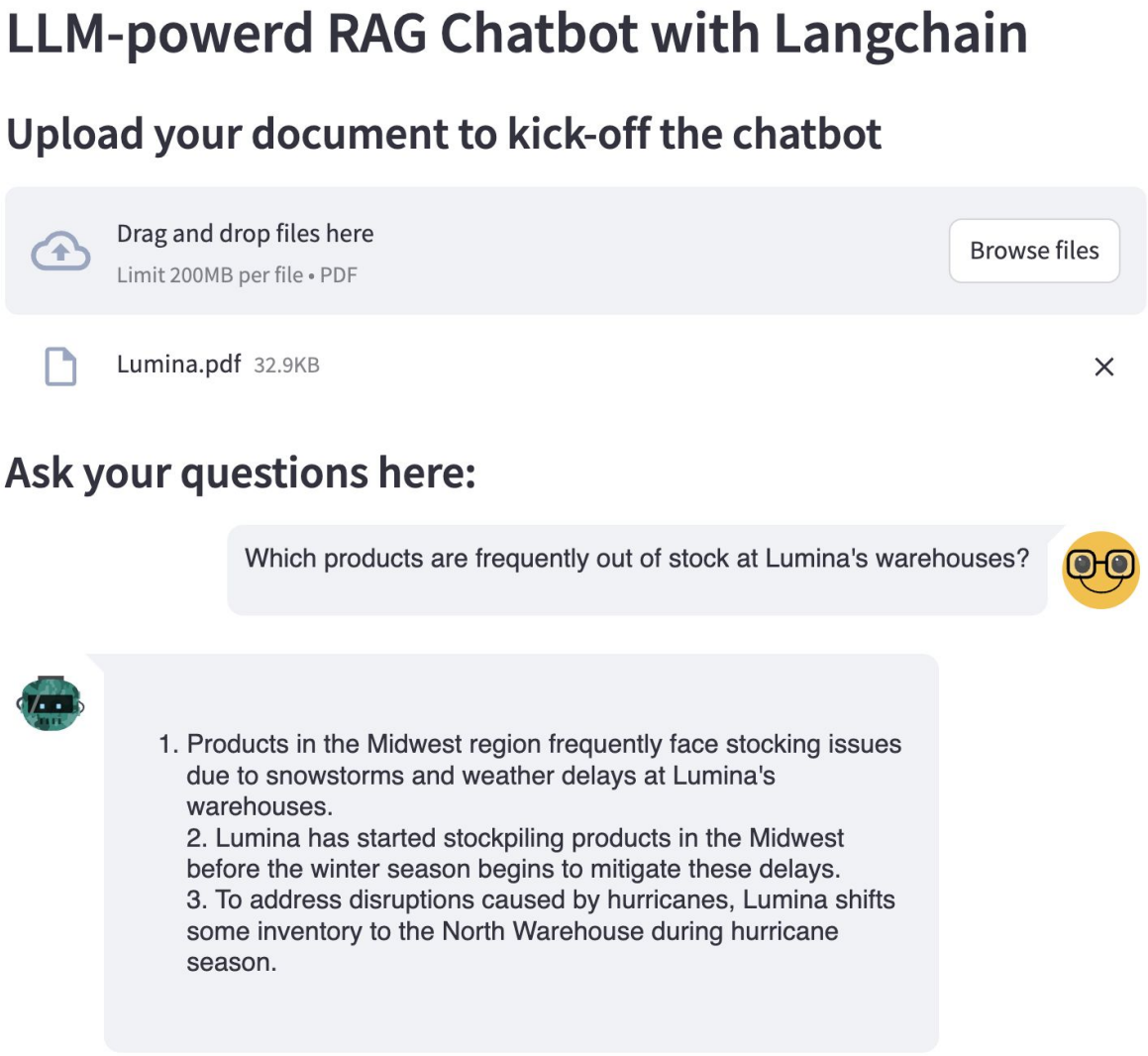
Let’s begin coding

Exercise 1

RAG chatbot using Lanchain and Vector database

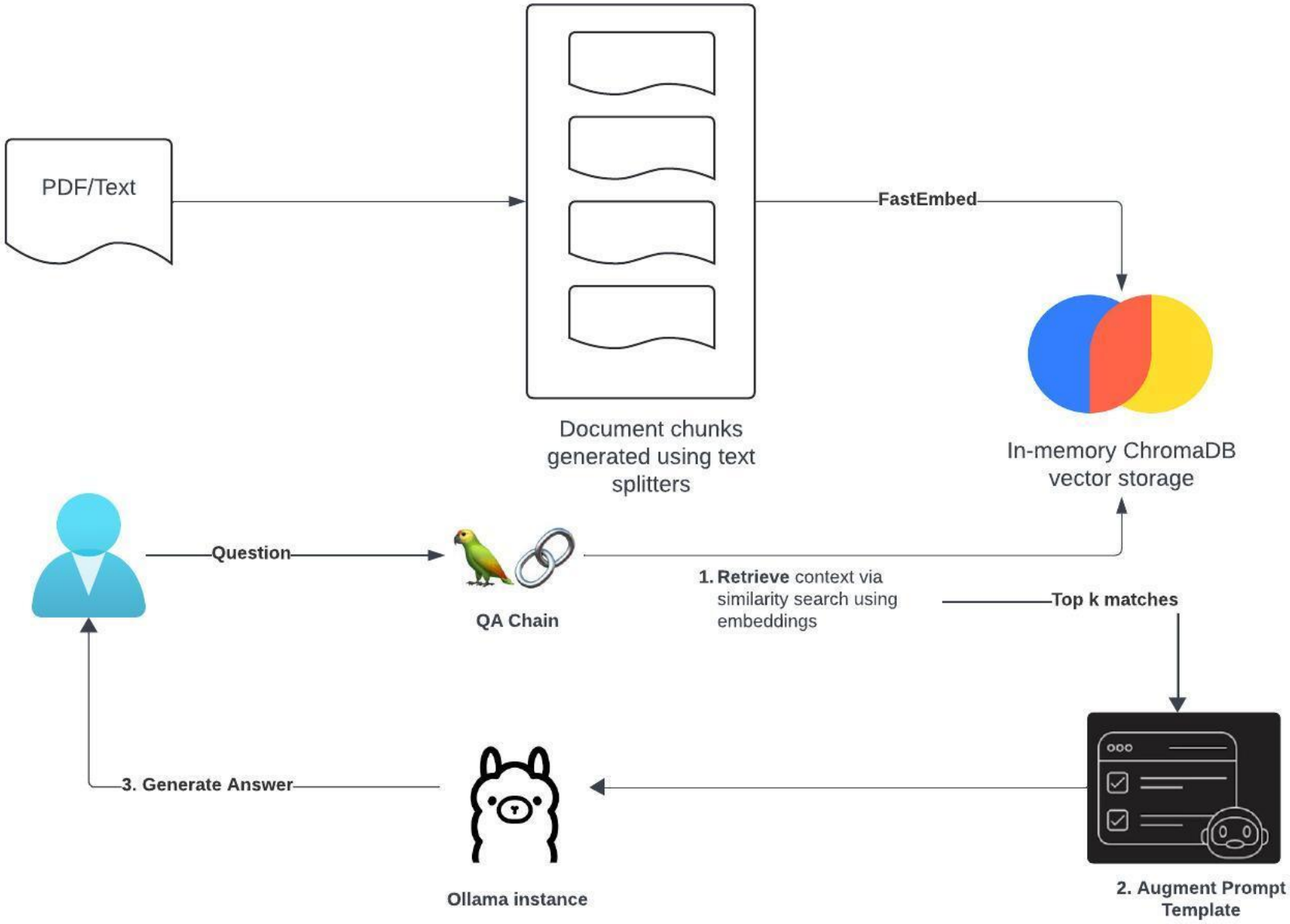
[Exercise 1 README]

Github Link:
https://tinyurl.com/ghc2024-aichatbot



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

RAG with Vector store



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

git checkout exercises

Let's build the RAG pipeline

The RAG pipeline is divided into the following steps which will be achieved from the following file

```
rag_pipeline.py
```

Step 1: Initiatlize Mistral model

Update the **init**() method - initialize Mistral using this API
https://api.python.langchain.com/en/latest/chat_models/langchain_community.chat_models.ollama.ChatOllama.html

```
    self.model = ChatOllama(model="mistral")
```

Step 2: Split the uploaded document into smaller chunks using text splitters

- Here, we are using the RecursiveCharacterTextSplitter

Add the following code in the method processDocument()

```
    self.text_splitter = RecursiveCharacterTextSplitter(chunk_size=200, chunk_overlap=20)
    document_chunks = self.text_splitter.split_documents(docs)
```

Step 3: Vectorize the document chunks using FastEmeddings and store in Chroma

```
    chroma_vector_store = Chroma.from_documents(documents=document_chunks, embedding=FastEmbedEmbeddings())
```

grace-hopper-2024-ai-chatbot / exercise-1-chatbot-rag-pdf / **rag_pipeline.py**

Code | Blame | 57 lines (42 loc) · 1.96 KB | Code 55% faster with GitHub Copilot

```
11  class ChatDocument:
12      vector_store = None
13      retriever = None
14      chain = None
15
16      def __init__(self):
17
18          ## Step 1: Initialize Mistral model
19          ## Add step 1 code here and remove 'pass'
20
21          pass
22
23      ## processDocument() method called from chat_ui.py
24      def processDocument(self, pdf_file_path: str):
25          ## Accepts a filepath
26          docs = PyPDFLoader(file_path=pdf_file_path).load()
27
28          ## Step 2: Initialize the text splitter to split the uploaded document into smaller chunks
29          ## Add step 2 code here
30
31          ## Filter out any complex data not supported by Chroma DB, before we pass it for vertorization
32          ## Please Comment out the following code
33          ## document_chunks = filter_complex_metadata(document_chunks)
34          ## print(document_chunks)
35
36          ## Step 3: Vectorize the document chunks using FastEmeddings and store in ChromaDB
37          ## Add step 3 code here
```

Step 4: Configure the Vector store Retriever for the type of search

```
self.retriever = chroma_vector_store.as_retriever(
        search_type="similarity_score_threshold",
        search_kwargs={
            "k": 10, ## return top k chunks
            "score_threshold": 0.50, ## with scores above this value
        },
    )
```

Step 5: Add a system prompt template using Langchain Prompts

```
self.prompt_from_template = PromptTemplate.from_template(
        """
        <s> [INST] You are an assistant for question-answering tasks. Use the following pieces of retrieved cont
        to answer the question. If you don't know the answer, just say that you don't know. Answer only as per w
        Use three sentences
         maximum and keep the answer concise. [/INST] </s>
        [INST] Question: {question}
        Context: {context}
        Answer: [/INST]
        """
    )
```

Step 6: Build a langchain conversion chain using prompt template and model with an output parser using LCEL

```
self.chain = ({"context": self.retriever, "question": RunnablePassthrough()}
                  | self.prompt_from_template
                  | self.model
                  | StrOutputParser())
```

```
## Step 4: Configures the Vector store Retriever class for the type of search
## Add step 4 code here

## Step 5: Add a system prompt template using Langchain Prompts
## Add step 5 code here

## Step 6: Build a chain of prompt template and model with an output parser using LCEL
## Add step 6 code here
```

ANITA B.ORG 2024
GRACE HOPPER CELEBRATION

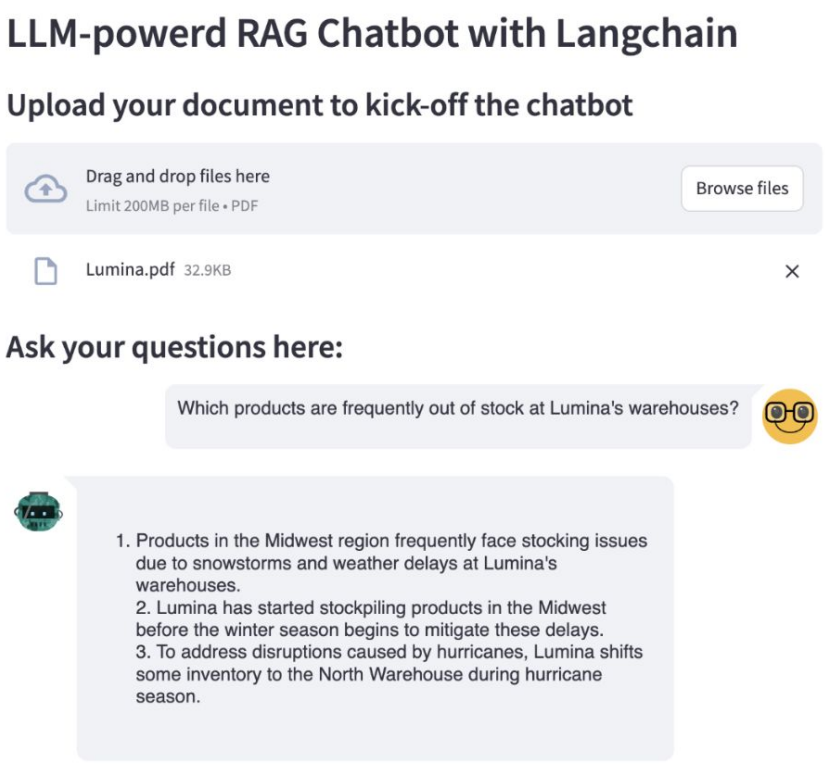
How to run this StreamLit app?

Execute

```
streamlit run chat_ui.py
```

Use the pdf file - Lumina.pdf (or Elena.pdf) in the folder and Ask questions about Lumina, the startup (or Elena)

Final output



If needed, you can run this command on the 'main' branch to see the final output

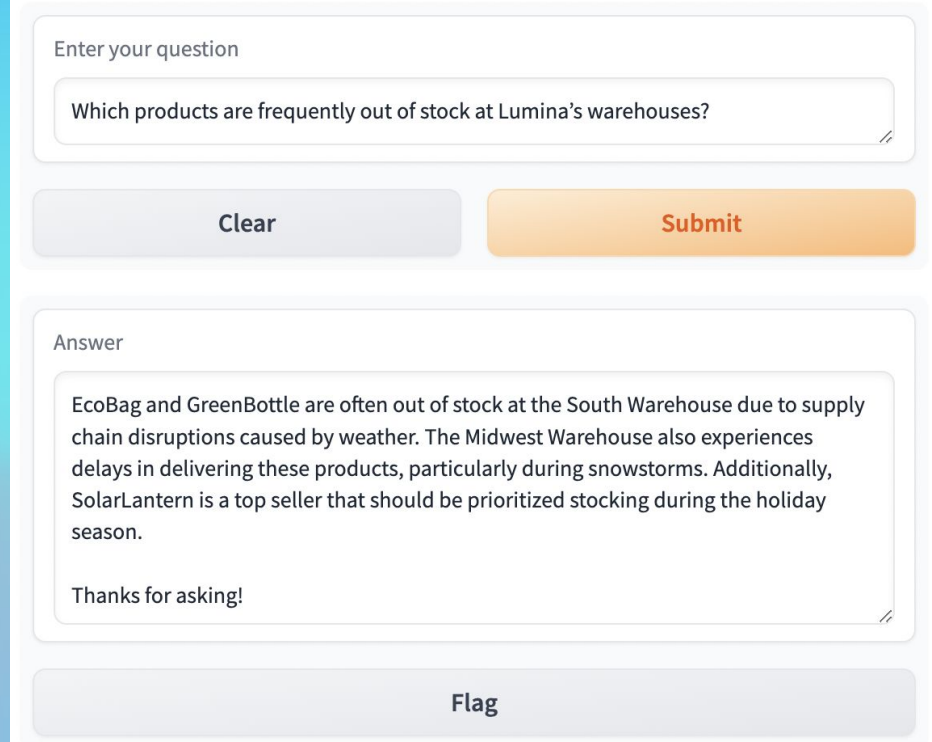
Exercise 2

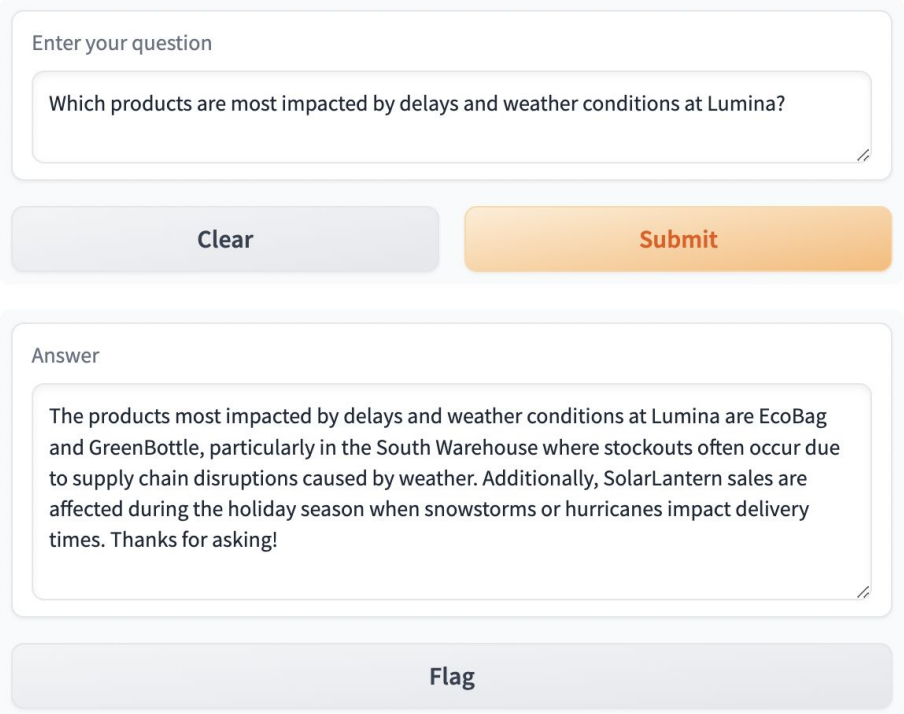
RAG chatbot using Langchain and Knowledge graph

[Exercise 2 README]

Github Link:

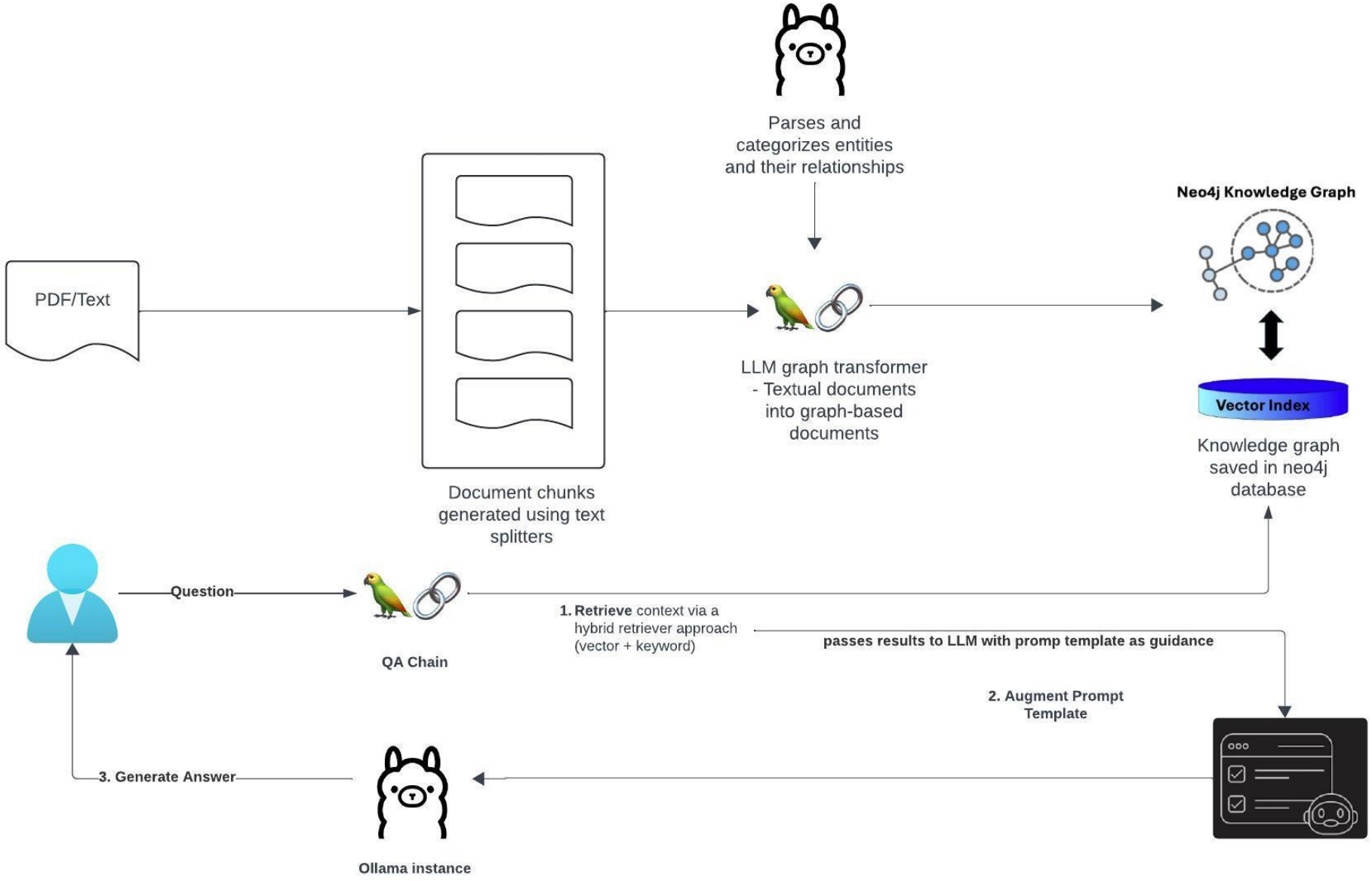
https://tinyurl.com/ghc2024-aichatbot





ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

RAG with Knowledge graph



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Neo4j Desktop Installation & Setup

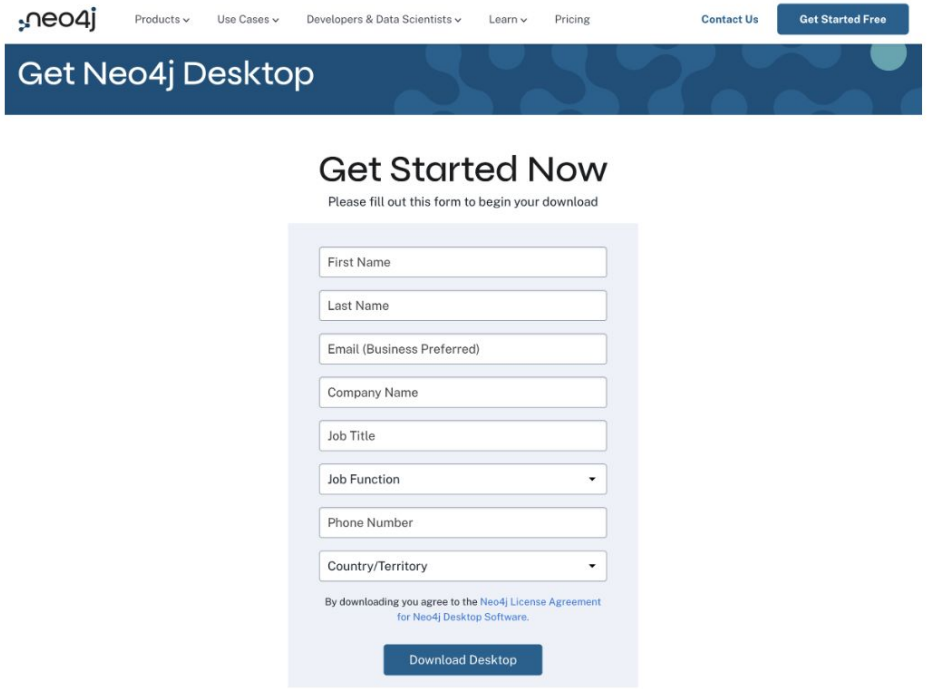
Pre-requisite 3: Neo4j Desktop Setup

The step-by-step process for installing and setting up Neo4j Desktop for this exercise are listed below. You can also watch this Neo4j Installation Guide Video for more reference: Neo4j Installation Guide Video

STEP-1: Installation and Local Setup of Neo4j Desktop Setup

1.1. Download the latest free version of Neo4j Desktop from here: https://neo4j.com/download/.

Fill the required form with basic details on the website to start the download.

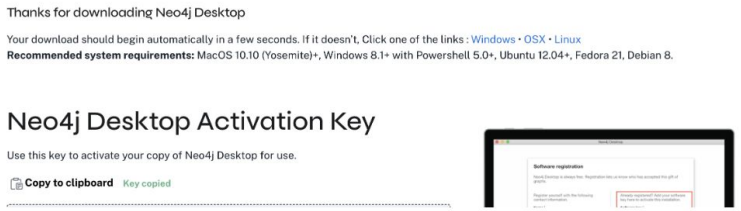


You can also view the various installation options for Neo4j on the Neo4j Deployment Center

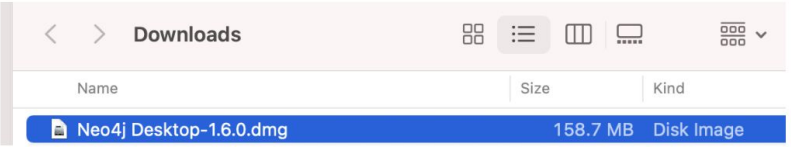
ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Neo4j Desktop Installation & Setup

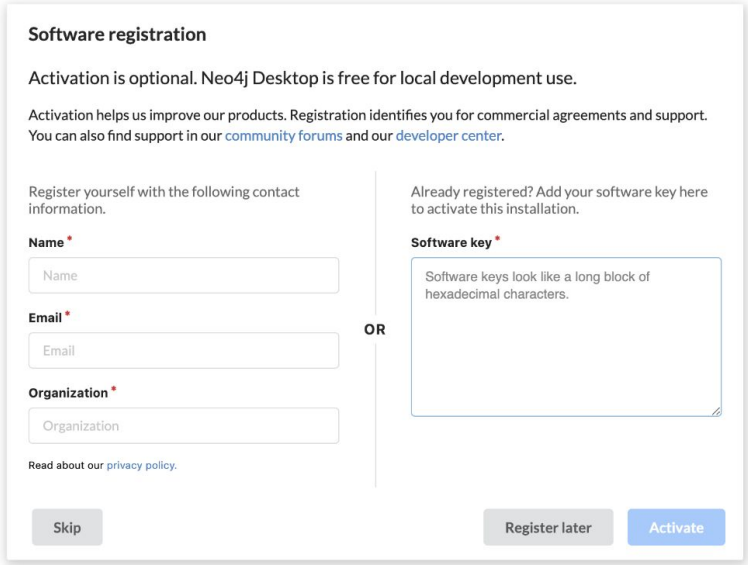
1.2. Copy the Neo4j Desktop Activation Key shown in the browser after downloading the Neo4j Desktop app.

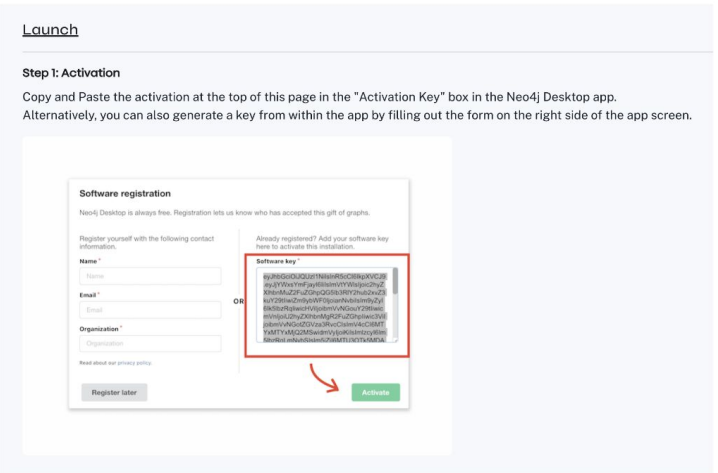


1.3. Open the downloaded .dmg installer file on your local for installing Neo4j Desktop.

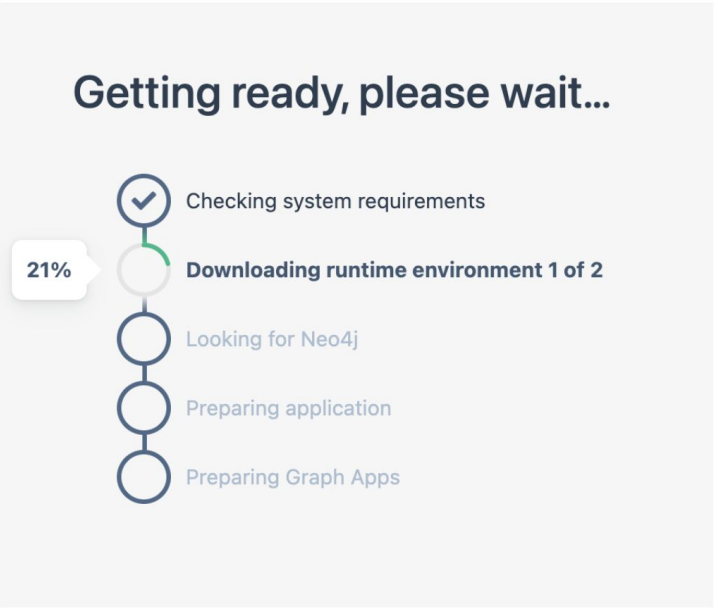


1.4. Paste the Neo4j Desktop Activation Key when asked during the installation steps.





1.5. Wait for few seconds or minutes for the installation to get completed.

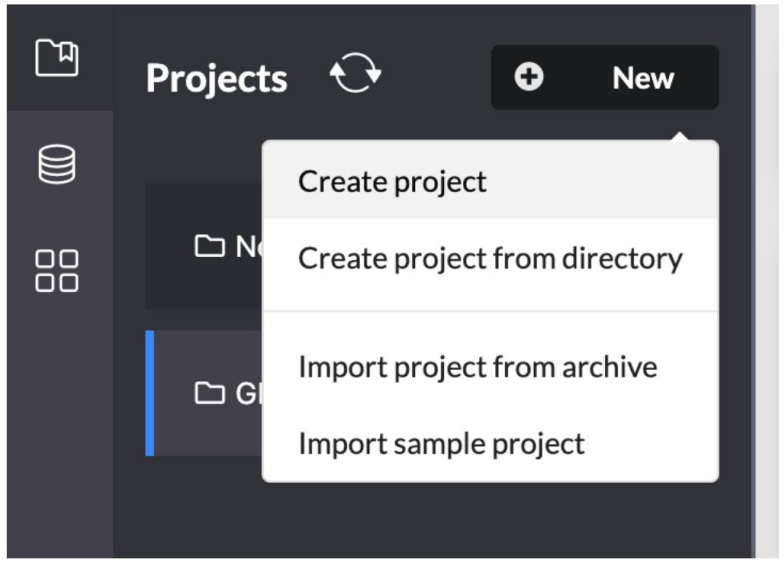


ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

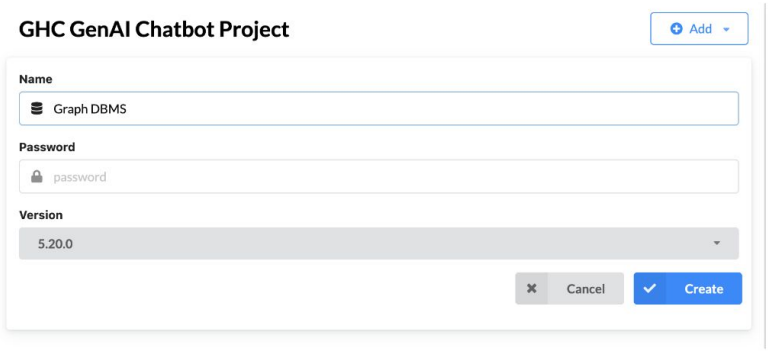
Neo4j Desktop Installation & Setup

STEP-2: Create a new Project and local DBMS in Neo4j Desktop

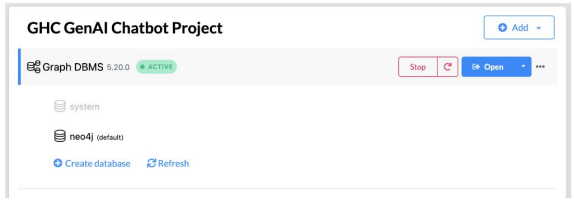
2.1. Create a new project in Neo4j Desktop.



2.2. Create a local DBMS (5.20.x) named Graph DBMS in this new project. Set the password and remember it.



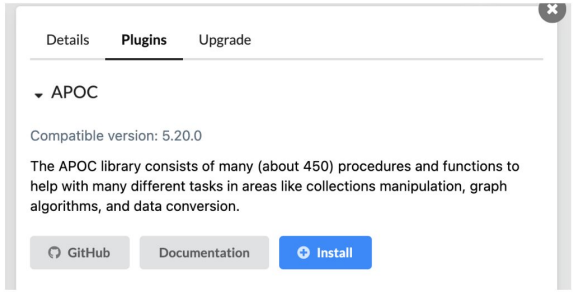
2.3. Click on Start the DBMS. It should start showing Active status once started along with the neo4j (default) database instance inside this DBMS.



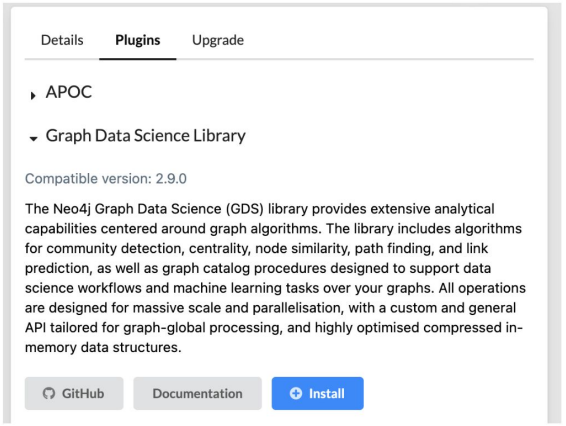
STEP-3: Install the required plugins

3.1. Click on the DBMS to open the right side panel. Click on the Plugins tab in the right side panel in Neo4j Desktop app.

3.2. Install the APOC (Awesome Procedures on Cypher) plugin. The APOC library consists of many functions to help with various different tasks in areas like collections manipulation, graph algorithms, and data conversion.



3.3. Install the GDS (Graph Data Science) plugin. The Neo4j Graph Data Science (GDS) library provides extensive analytical capabilities centered around graph algorithms.



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Neo4j Desktop Installation & Setup

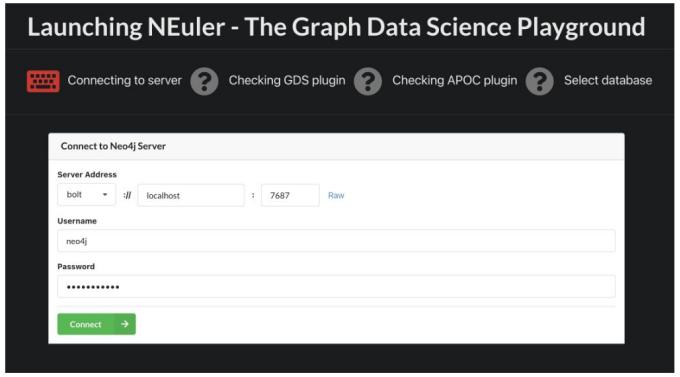
STEP-4: Connecting to the DBMS in the Graph Data Science Playground

4.1. Now, click on the Graph Apps option in the left side panel in the Neo4j Desktop app.

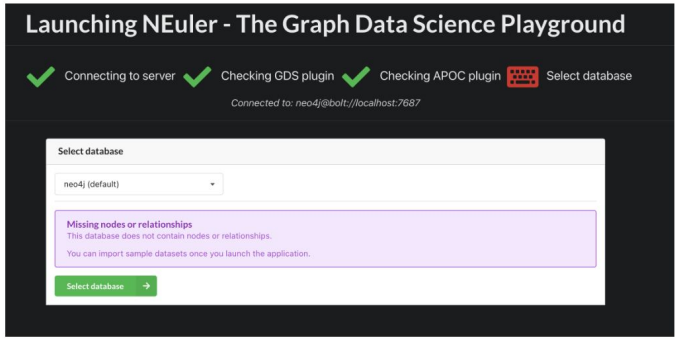
4.2. Click on the Graph Data Science Playground to launch it.

4.3. The Graph Data Science Playground (NEuler) will be launched in a new window. It is a project of Neo4j Labs and is an excellent way to explore smaller graphs.

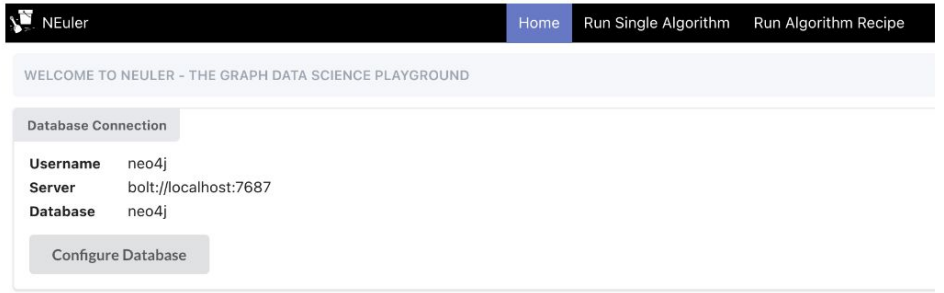
4.4. Ensure all checks are passing and connect to the local neo4j server.



4.5. Select the pre-selected neo4j (default) database instance



4.6. Kudos, the local Neo4j Database Connection is done, you're good to go ahead.



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Step 1

Step 1: Initialize Neo4j

- Install Neo4j Desktop using above installation steps and replace the password value in code.
- Create Neo4j database wrapper for performing the graph operations. About Neo4jGraph

```
graph = Neo4jGraph()
```

```
# Refer to the Neo4j Desktop Installation & Setup Steps in the Readme.
os.environ["NEO4J_URI"] = "bolt://localhost:7687"
os.environ["NEO4J_USERNAME"] = "neo4j"
os.environ["NEO4J_PASSWORD"] = "Password" # Replace with your neo4j password value here

## Step-1: Create Neo4j database wrapper for performing the graph operations.
## Add step-1 code here.
```

```
try:
    # Read the text from the file
    with open(file_path, "r", encoding='utf-8') as file:
        text = file.read()

    # Initialize RecursiveCharacterTextSplitter with desired parameters
    # Read more about it here: https://sj-langchain.readthedocs.io/en/latest/text_splitter/langchain.text_splitter.RecursiveCharacterTextSplitter.html
    # RecursiveCharacterTextSplitter recursively breaks down text into smaller chunks for easier processing and retrieval.
    # This is the recommended text splitter for generic text.
    # It is parameterized by a list of characters, the default list is ["\n\n", "\n", " ", ""]. It tries to split on them in order until the chunks are small enough.
    # This has the effect of trying to keep all paragraphs (and then sentences, and then words) together as long as possible, as those would generically seem to be the strongest semantically related pieces of text.
    # Adjust chunk_size and chunk_overlap according to your needs.
    # The chunk size should be at least as large as the average length of your queries.
    # The chunk overlap should be smaller than the chunk size to maintain context.
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=512,  # Adjust chunk size as needed
        chunk_overlap=24  # Adjust overlap to maintain context
    )

    # Convert the formatted text into a list of Document objects
    texts = []

    # Docstores are classes to store and load Documents.
    # The Docstore is a simplified version of the Document Loader.
    # https://python.langchain.com/v0.2/api_reference/community/docstore.html
    document = Document(page_content=text)
    texts.append(document)

    # Split the text into chunks
    documents = text_splitter.split_documents(texts)
    # Print the split chunks
    for i, chunk in enumerate(documents):
        print(f"Chunk {i+1}:\n{chunk}\n")

# Print an error message if the file is not found.
except FileNotFoundError:
    print(f"Error: File '{file_path}' not found.")
```

ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Step 2

Step 2: Convert textual documents into graph-based documents using LLMGraphTransformer

- Create an instance of LLMGraphTransformer, it converts textual documents into graph-based documents using LLM. About LLMGraphTransformer

  It includes passing below parameters:

  - llm: Pass the Llama3 instance running using Ollama
  - allowed_nodes: Specifies which node types are allowed in the graph. Defaults to an empty list, allowing all node types.
  - allowed_relationships: Specifies which relationship types are allowed in the graph. Defaults to an empty list, allowing all relationship types.

```
llm_transformer = LLMGraphTransformer(
    llm=llm,
    allowed_nodes=["Company", "Person", "Product", "Supplier", "Warehouse", "Store", "Shipment", "Customer", "Ex
    allowed_relationships=["WORKS_FOR", "CREATES", "SUPPLIES", "STOCKS", "DELIVERS_TO", "LOCATED_AT", "AFFECTED_I
)
```

- Extract graph data by converting a sequence of documents into graph documents. About LLMGraphTransformer.convert_to_graph_documents()

```
graph_documents = llm_transformer.convert_to_graph_documents(documents)
print("graph documents", graph_documents)
```

```
# Instantiating our model object with relevant params for Chat Completion.
# Running LLaMA3.1-8b LLM locally using Ollama.
# https://api.python.langchain.com/en/latest/chat_models/langchain_community.chat_models.ollama.ChatOllama.html
llm = ChatOllama(
    model="llama3.1:8b", # LLM Model running locally using Ollama
    temperature=0.1, # Temperature parameter for LLM model, ranges from 0 to 1, controls the randomness of the output.
)

## Step-2: Convert textual documents into graph-based documents using LLMGraphTransformer
## Add step-2 code here and uncomment the below print lines to see the generated graph details.


# # Print the nodes and relationships of the generated graph
# print(f"Nodes:{graph_documents[0].nodes}")
# print("----------------------------------------------------------------------")
# print(f"Relationships:{graph_documents[0].relationships}")

# # Print all the nodes and relationships of the generated graph
# for graph_doc in graph_documents:
#     print("\nNodes:")
#     for node in graph_doc.nodes:
#         print(f"Id: {node.id}, Type: {node.type}")
#     print("\nRelationships:")
#     for relationship in graph_doc.relationships:
#         print(f"Source: {relationship.source}, Target: {relationship.target}, Type: {relationship.type}")
```

ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Steps 3 and 4

```
## Step-3: Store to neo4j
## Add step-3 code here.

## Step-4: Initialize and return a Neo4jVector instance from existing graph using OllamaEmbeddings.
## Add step-4 code here.
```

Step-3: Store to neo4j

- Store to neo4j using add_graph_documents(), this method constructs nodes and relationships in the graph based on the provided GraphDocument objects. About Neo4jGraph add_graph_documents()

```
graph.add_graph_documents(
   graph_documents,
   baseEntityLabel=True,
   include_source=True
)
print("Documents successfully added to Graph DataBase")
```

Step-4: Initialize and return a Neo4jVector instance from existing graph using OllamaEmbeddings

- Instantiate the Ollama embedding model and generate embeddings using the locally running LLaMA3.1-8b About OllamaEmbeddings

```
local_embeddings = OllamaEmbeddings(model="llama3.1:8b")
```

- Initialize and return a Neo4jVector instance from existing graph. This method initializes and returns a Neo4jVector instance using the provided parameters and the existing graph. It validates the existence of the indices and creates new ones if they don't exist. About Neo4jVector.from_existing_graph()

```
vector_index = Neo4jVector.from_existing_graph(
   embedding=local_embeddings,
   search_type="hybrid",
   node_label="Document",
   text_node_properties=["text"],
   embedding_node_property="embedding"
)
```

Step 5

```
## Step-5: Create a RetrievalQA chain using custom prompt template and the Neo4jVector as the retriever.
## Add step-5 code here.
```

Step-5: Create a RetrievalQA chain for question-answering

- Create custom prompt template. A prompt template consists of a string template, it can contain the system prompt, human question and context.

```
template = """Use the following pieces of context to answer the question at the end.
If you don't know the answer, just say that you don't know, don't try to make up an answer.
Use three sentences maximum and keep the answer as concise as possible.
Always say "thanks for asking!" at the end of the answer.
{context}
Question: {question}
Helpful Answer:"""
```

- Create a PromptTemplate instance with the custom prompt template for the language model. About PromptTemplate

```
QA_CHAIN_PROMPT = PromptTemplate(input_variables=["context", "question"], template=template)
```

- Create a RetrievalQA Chain for question-answering against an index using the Neo4jVector as the retriever. About RetrievalQA

```
qa_chain = RetrievalQA.from_chain_type(
    llm,
    retriever=vector_index.as_retriever(),
    chain_type_kwargs={"prompt": QA_CHAIN_PROMPT}
)
```

ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Step 6

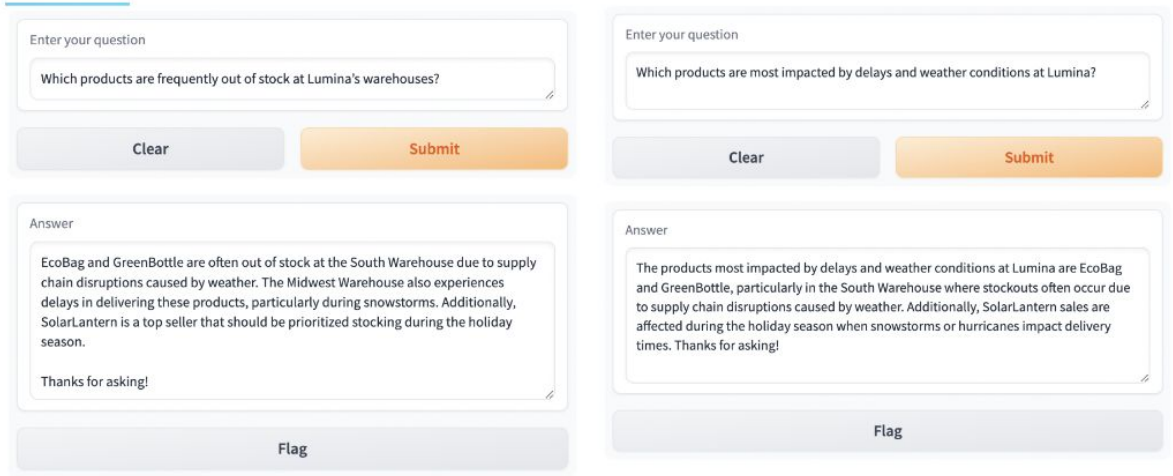
```python
# Define the function for querying document details
def query_document_details(query):
    if query is None:
        return "Error: Query cannot be None"
    try:
        result = qa_chain({"query": query})
        return result["result"]
    except Exception as e:
        return f"Error: {str(e)}"

# Create a Gradio interface
# https://www.gradio.app/guides/the-interface-class
interface = gr.Interface(
    fn=query_document_details,        # Function to call
    inputs=gr.Textbox(label="Enter your question"),  # Input textbox as Gradio component to be used for the input
    outputs=gr.Textbox(label="Answer")   # Output textbox as Gradio component to be used for the output
)

# Launch the interface
interface.launch()
```

Step-6: Execute the Exercise-2 code

Run the below command to execute the exercise-2 code.

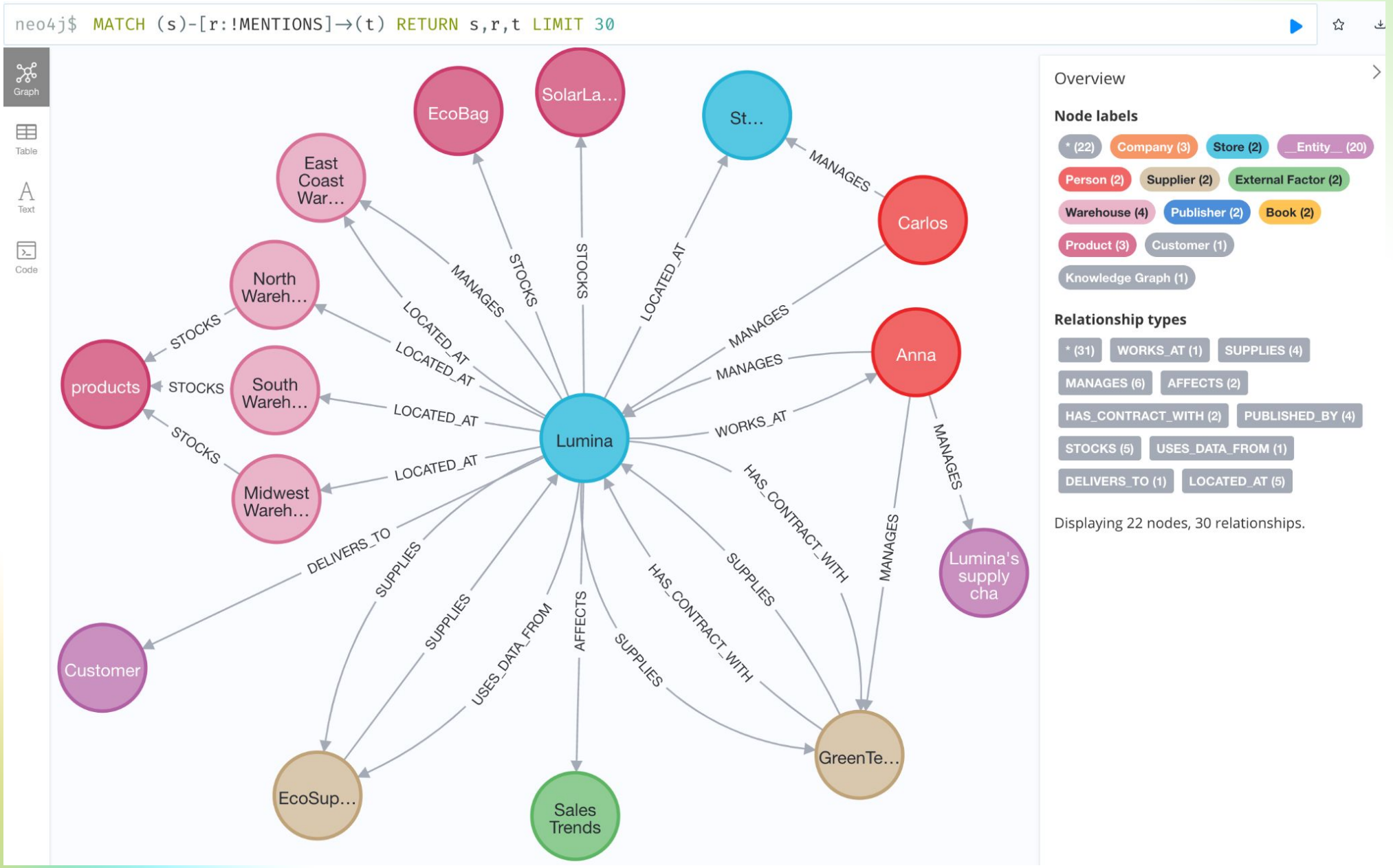
```
python3 knowledge-graph-rag.py
```

Final output



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

Neo4j Knowledge Graph Visualization



ANITA B.ORG 2024 GRACE HOPPER CELEBRATION

What's next?

You can further experiment with:

- Experimenting with this example on other data sources
- RAG on multi-modal data
- Adding few-shot examples in the prompt
- Adding memory to the conversation chain
- Add a re-ranker to improve results

Learning resources:

- DeepLearning.AI Courses
- Neo4j GraphAcademy
- Langchain Official Documentation

THANK YOU

Networking Google Sheet - to stay connected!

Reeti Sarup

Senior Software Engineer | Intuit

Rishu Kaur

Group Engineering Manager | Intuit