Assignment 4.2: Building a RAG Pipeline with Airflow

Due:21st March 2025, 03:59 PM EST

Required Attestation and Contribution Declaration

WE ATTEST THAT WE HAVEN'T USED ANY OTHER STUDENTS' WORK IN OUR ASSIGNMENT AND ABIDE BY THE POLICIES LISTED IN THE STUDENT HANDBOOK

Contribution:
Member 1: 33%
Member 2: 33%
Member 3: 33%

Links to GitHub tasks and tasks owned by each member

Instructions:

You are working at a startup that aims to build an AI-powered information retrieval application that processes unstructured data sources such as PDFs and web pages. Your task is to design and implement a Retrieval-Augmented Generation (RAG) pipeline using **Airflow**, ensuring modularity and extensibility for future applications.

Your team is given the following requirements:

1. **Data:**
    - Get the NVIDIA quarterly reports for the past 5 years
2. **Implement a Data Pipeline:**
    - Use **Apache Airflow** to orchestrate data ingestion, processing, and retrieval workflows.
3. **Parsing PDFs. Implement 3 strategies**
    - Build upon Assignment 1's extraction capabilities.
    - Use **Docling** for parsing PDFs.
    - Explore **Mistral OCR** (https://mistral.ai/news/mistral-ocr) for improved text extraction.
4. **Building the RAG Pipeline:**
    - Implement a naive RAG system without a vector database, computing embeddings and cosine similarity manually.
    - Integrate with **Pinecone**
    - Integrate with ChromaDB for advanced retrieval.

    - Implement at least **three chunking strategies** to optimize retrieval.
    - Implement **hybrid search** so that you can query a specific quarter's data only to get context
5. **Testing & User Interface:**
    - Develop a **Streamlit** application allowing users to:
        - Upload PDFs.
        - Select PDF parser (Docling, Mistral OCR, etc.).
        - Choose RAG method (manual embeddings, Pinecone, ChromaDB).
        - Select chunking strategy.
        - Select the quarter/quarters data to be used to answer the query
    - Use **FastAPI** to connect with the RAG pipeline and return relevant document chunks based on user queries.
    - Use your preferred **LLM** to process and generate responses.
6. **Deployment:**
    - Create **two Docker pipelines**:
        - **Airflow pipeline** for data ingestion, processing, and retrieval.
        - **Streamlit + FastAPI pipeline** for user interaction and querying.

Submission:

GitHub Repository:

- Include a **Project Summary** and **Proof of Concept (PoC)**.
- Use **GitHub Issues** to track tasks.
- Include **diagrams**, a **fully documented CodeLab**, and a **5-minute demo video**.
- Provide a link to the **hosted application** and backend services.

Documentation:

- Comprehensive **README.md** with project setup and usage instructions.
- **AIUseDisclosure.md** detailing AI tools used and their purpose.

Resources:

- **Docling GitHub Repository**
- **Mistral OCR Documentation**
- **Apache Airflow Documentation**
- **Pinecone & ChromaDB Documentation**
- **FastAPI & Streamlit Documentation**
- **AWS S3 Best Practices**

Evaluation Criteria:

Category	Weightage
Data Extraction & Parsing	**25%**
RAG Implementation & Chunking Strategies	**40%**
Streamlit UI & FastAPI Integration	**15%**
Deployment & Dockerization	**10%**
Documentation & Presentation	**10%**