Take-Home Challenge: Building a RAG-based Q&A Chatbot

In this task, you are required to build a simple web application that allows users to upload documents and perform question-answering (Q&A) based on the upload content. The Q&A chatbot should utilize **Retrieval-Augmented Generation (RA** to enhance the AI's responses using relevant information from the uploaded documents. The core functionalities include document upload, text preprocessing, embedding generation, vector storage, and dynamic querying.

Stack

• **Django:** Backend for handling document upload, preprocessing, embeddings, and database interactions

• **React:** Frontend for document upload interface and chatbot UI

• **Postgres + pgvector:** For storing document embeddings and enabling similarity-based retrieval

• **Docker:** For containerization

• **OpenAI:** For embedding generation and LLM-based response generatio

• **LangChain/LlamaIndex:** For integrating the embedding and retrieval workflow

Instructions

1. Document Upload and Preprocessing

Create a Django backend endpoint to allow users to upload documents (support at least PDF format).

Once a document is uploaded, extract the text.

2. Embedding Generation

Use **OpenAI's** text-embedding-ada-002 or a similar embedding model to convert the cleaned text into vector representations (embeddings).

Break the extracted text into smaller chunks (e.g., sentences or paragraphs) to fit within token limits before generating embeddings.

3. Vector Database

Store the generated embeddings in **Postgres with pgvector** extension for vector similarity search.

Ensure the embeddings are indexed for efficient similarity-based retrieval (cosine similarity or similar distance metric).

4. Query-Based Retrieval and Augmented Response

Create an endpoint in Django that takes a user query, embeds the query using the same embedding model, and searches the vector database for the most relevant text chunks.

Dynamically construct a prompt that includes the retrieved text along with the user query, and send it to **OpenAI**'s LLM to generate a final response

Implement a fallback mechanism that generates a response without augmentation if no relevant text is found in the vector database.

5. React Frontend

Build a React-based frontend where users can:

Upload PDFs to the knowledge base.

Ask questions in a chatbot-style interface and receive responses that may include relevant content from the uploaded documents.

6. Containerization

Use docker-compose to containerize and manage the application's backend frontend, and Postgres database with the pgvector extension.

Ensure that when the containers are started, necessary migrations and database setup are automatically performed.

Deliverables

A fully functional web application that satisfies the above requirements.
A README.md file with instructions on how to set up and run the application using Docker.

Once you have completed the project, please email your solution, including all necessary files and documentation, to dev@neuralityhealth.ai

Happy coding!