RAG Bot Architecture and Approach

1. Model Architecture:-

This Retrieval-Augmented Generation (RAG) bot architecture consists of three core components:

1. Document Ingestion and Processing: PDFs are uploaded and processed for text extraction.
2. Vector Store for Retrieval: The text is chunked, embedded using Google Generative AI embeddings, and stored in a Pinecone vector database for retrieval.
3. Conversational QA Generation: The RAG bot performs similarity search over the vectorized chunks to retrieve relevant documents and uses Google Generative AI to generate answers fromthe retrieved context.

2. Approach to Retrieval:-

The retrieval process ensures that relevant sections of the documents are identified before a response is generated. The steps include:

1. Vectorization: After text extraction, the PDF text is split into chunks (10,000 characters with an overlap of 1,000). These chunks are then embedded into 768-dimensional vectors using Google Generative AI Embeddings (model="models/embedding-001").

2. Storing in Pinecone: The generated vectors are stored in Pinecone, a high-performance vector database, for fast and scalable similarity search. Pinecone is initialized with serverless cloud specs to allow easy index management and scaling.

3. Similarity Search: Upon receiving a user question, the system uses Pinecone's similarity_search function to find the top document chunks most similar to the user's query.

These retrieved chunks are then passed to the response generation stage.

3. Generative Responses:-

Once relevant chunks are retrieved from Pinecone, they are passed to Google Generative AI (specifically the gemini-pro model) to generate a detailed and accurate response based onthe following key components:

1. QA Chain: I use Langchain's load_qa_chain method to load a chain capable of handling question-answering tasks. A custom prompt template is applied to ensure high-quality,context-driven answers.

2. Prompt Template: The prompt used ensures that responses are accurate and derived from the retrieved context. If an answer cannot be found in the context, the model responds with 'Answer is not available in the context.'

3. Answer Generation: The Google Generative AI model (gemini-pro) is then used to generatean answer. The answer is derived from the retrieved documents and displayed in the UI via Streamlit.

4. Explanation of Decisions and Challenges Faced:-

- Choosing Google Generative AI for Embeddings: I used Google Generative AI Embeddings (models/embedding-001) for high-quality vector representation. These embeddings were selected for their effectiveness in capturing semantic meaning, which is essential for accurate retrieval.
- Using Pinecone for Retrieval: Pinecone was chosen for its scalable and efficient vector-based similarity search. It allows quick and accurate retrieval of the most relevant textchunks based on the query.
- Streamlit for User Interface: I leveraged Streamlit for its simplicity in building and deploying an interactive UI. The UI allows users to upload PDFs, process them, and ask questions, displaying responses in real-time.

Challenges and Solutions:

1. Embedding Model Selection: Finding an appropriate embedding model that balances performance and accuracy was critical. The selected model (embedding-001) was well-suited for large document chunks and handled semantic understanding effectively.

2. Large Document Handling: PDFs often contain large amounts of text, requiring chunking to handle them efficiently. Setting the right chunk size (10,000 characters with a1,000-character overlap) ensured a good balance between retrieval granularity andperformance.

3. Context Length Limits: Generative models have limitations on context length. By carefully retrieving and feeding relevant chunks from Pinecone, we ensured that only the most pertinent data was passed to the model, avoiding token overflow.

4. Real-time Response Speed: Generating responses in real-time was crucial. I addressed this by optimizing the embedding and retrieval process to minimize latency.

5. How to Use the RAG Bot:-

1. Upload PDFs: Users upload PDF files in the sidebar.

2. Process PDFs: Click 'Submit 2 Process' to process the uploaded PDFs.

3. Query Data: Once the PDFs are processed, users can select a PDF from a dropdown list and ask questions related to its content.

4. Receive Response: The bot retrieves the relevant document chunks, passes them to Google Generative AI, and displays the answer on the screen.