

Interactive QA Bot Documentation

1. Model Architecture

1.1 Overview

The Interactive QA Bot utilizes a Retrieval-Augmented Generation (RAG) model architecture that combines information retrieval with generative responses. The architecture is built on three primary components:

- **Embedding Model:** Generates dense vector representations of the text chunks using SentenceTransformer.
- **Vector Database:** Stores and retrieves embeddings using Pinecone, allowing for efficient similarity searches.
- **Generative Model:** Utilizes the Cohere API to generate context-aware answers based on retrieved text.

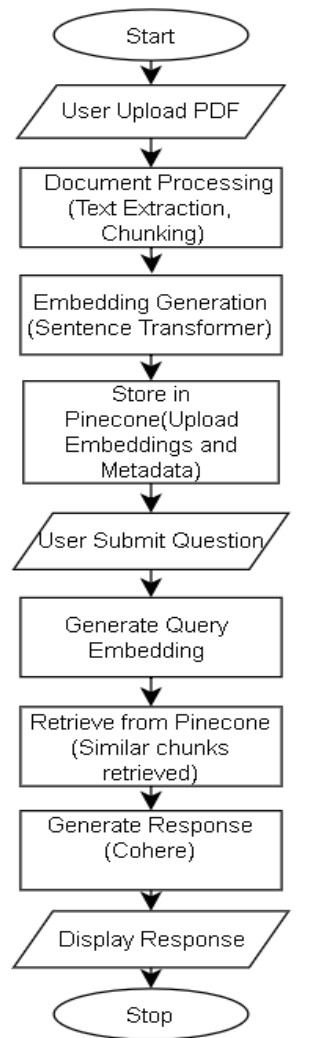


Figure 1.1 Overview

1.2 Approach to Retrieval

- **Text Extraction:** The bot extracts text from uploaded PDF documents using the pdfplumber library.
- **Chunking:** The extracted text is split into smaller, manageable chunks to facilitate efficient embedding generation and retrieval.
- **Embedding Generation:** Each chunk is converted into an embedding using the all-MiniLM-L6-v2 model.
- **Vector Search:** When a user submits a query, the bot generates an embedding for the query and retrieves the top k most similar chunks from the Pinecone vector database based on cosine similarity.

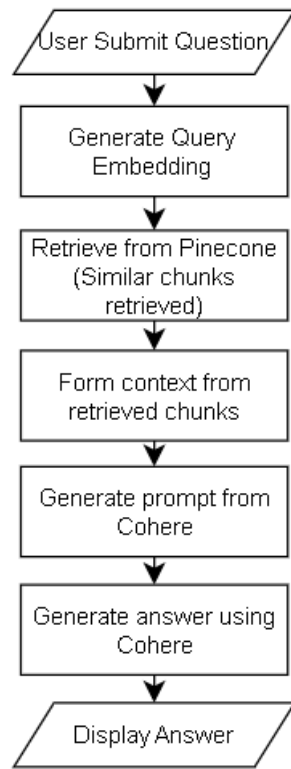



Figure 1.2 Text Retrieval

1.3 Generative Response Creation


- **Context Concatenation:** The bot concatenates the text of the retrieved chunks into a single context string.
- **Prompt Generation:** A prompt is created to instruct the Cohere model to generate an answer based on the provided context and the user's query.
- **Answer Generation:** The response from the Cohere model is processed and returned as the final answer.

2. User Instructions

2.1 Uploading Files

1. Open the Streamlit app at the provided public URL.
2. Click on the "  Upload a PDF document" button.
3. Select a PDF file from your device and upload it.

2.2 Asking Questions

1. Once the PDF is processed, enter your question in the text input box labeled "  Ask any question about the document".
2. Click on the "Submit" button.

2.3 Viewing Bot Responses

- The bot's responses will appear below the query input area, along with the chat history of your interactions.

3. Example Interactions Demonstrating the Bot's Capabilities

On the Chatbot Interface I uploaded the assignment file

Interaction 1:

1. **User:** What is the main purpose of developing a RAG model for a QA bot?
2. **Bot:** The main purpose of developing a Retrieval-Augmented Generation (RAG) model for a Question Answering (QA) bot is to create a system that can efficiently retrieve relevant information from a dataset and generate coherent answers to questions. The RAG model combines document retrieval techniques with generative modeling to provide accurate and contextually relevant responses.

Interaction 2:

1. **User:** What is the role of the vector database in this project?
2. **Bot:** The vector database, such as Pinecone, is used to store and retrieve document embeddings efficiently as part of the RAG-based model for the QA bot.

Interaction 3:

1. **User:** What are the deliverables expected from this project?
2. **Bot:**
 - A deployed Q&A system that can process PDF documents and provide answers to user queries.
 - Source code for the notebook and interface.
 - A fully functional Colab notebook.
 - Documentation on the pipeline and deployment instructions.
 - A simple frontend interface that allows users to upload PDF documents and ask questions.
 - An integrated backend that can process PDFs, store document embeddings, and provide real-time answers.
 - The system should handle multiple queries efficiently and provide accurate, contextually relevant responses.

4. Approach, Decisions, Challenges, and Solutions

4.1 Approach

- **User-Friendly Interface:** The goal was to make the interface simple so anyone could upload a document and ask questions without needing technical knowledge.
- **Modular Design:** The project was broken into small parts (like text extraction, embedding, retrieval, and answer generation) to make it easy to develop, test, and improve each part separately.
- **Rapid Prototyping:** I used tools like Google Colab, Pinecone, and Cohere to quickly test and improve the system without setting up a lot of infrastructure.
- **Scalability:** The design ensures that the system can handle larger documents and more queries as it grows.

4.2 Decisions

- **Embedding Model:** I chose SentenceTransformer because it provides a good balance of speed and accuracy in understanding the meaning of text.
- **Pinecone:** Pinecone was selected for storing and retrieving vector embeddings because it handles large-scale searches quickly and efficiently.

- **Cohere:** Cohere's command-r-plus model was chosen for generating answers because it performs well in answering questions based on given context.
- **Chunking:** I broke long texts into smaller chunks to stay within the limits of the embedding and generative models.

4.3 Challenges

- **Retrieval Accuracy:** Making sure the retrieved information was relevant to the query was sometimes difficult, especially with long documents.
- **Context Limits:** Both Pinecone and Cohere have limits on how much text they can handle at once, so managing this without losing important information was a challenge.
- **Performance:** Processing large documents in real time while keeping the system fast and responsive was difficult.

4.4 Solutions

- **Improved Chunking:** Refined how the text was broken up into chunks, keeping important information together, which made the retrieval more accurate.
- **Prompt Engineering:** It helped in generating more accurate and context-aware responses from the generative model.

- **Combining Chunks:** Combined the most relevant chunks into one piece of context for the generative model, ensuring it didn't lose important information.
- **Optimizing Performance:** To speed things up, I cached embeddings for documents, so we didn't have to recompute them, and we processed text in batches.

5. Pipeline and Dockerization

5.1 Pipeline Overview

1. **Text Extraction:** Extract text from PDF using pdfplumber.
2. **Text Chunking:** Split the text into manageable chunks.
3. **Embedding Generation:** Generate embeddings for each text chunk using SentenceTransformer.
4. **Data Storage:** Store embeddings in Pinecone for retrieval.
5. **User Interaction:** Use Streamlit for a web interface to allow user uploads and queries.
6. **Response Generation:** Use Cohere's API to generate responses based on the retrieved text.

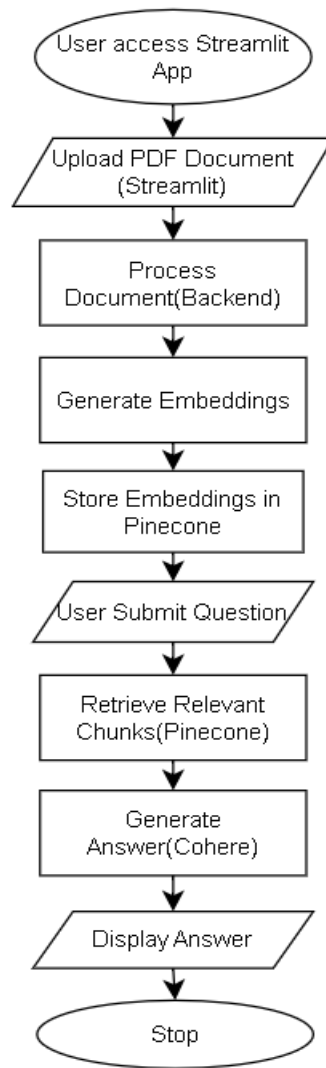


Figure 5.1 Pipeline cloud

5.2 Dockerization

To containerize the Interactive QA Bot application, the following steps were followed:

1. **Dockerfile Creation:** A Dockerfile was created to define the environment for the application. This file specifies the base image, installs necessary dependencies, and sets up the working directory.
2. **Building the Docker Image:** To build the Docker image, navigate to the directory containing the Dockerfile and run the following command:

```
docker build --no-cache -t interactive-qa-bot-app .
```

3. **Running the Docker Container:** After building the image, run the Docker container using the command below, which maps the local port to the container's port:

Create an account on Cohere and Pinecone to retrieve api keys and follow these steps:

```
docker run -p 8501:8501 \  
-e COHERE_API_KEY="your-cohere-api-key" \  
-e PINECONE_API_KEY="your-pinecone-api-key" \  
interactive-qa-bot-app
```

4. **Accessing the Application:** Once the container is running, open your web browser and navigate to **http://localhost:8501** to access the Streamlit application.

6. Execution and Deployment

6.1 Using Google Colab

1. Open the provided Jupyter Notebook file (`RAG_Q&A_Bot.ipynb`) in Google Colab.
2. For accessing Pinecone, Ngrok and Cohere APIs, you will need to provide your own API keys

Create an account on Pinecone, Ngrok and Cohere to get your own API keys and substitute their values in respective fields in the notebook.

****NOTE**** ngrok is essential in Google Colab for making your Streamlit application accessible over the internet.

4. Run all the cells
5. After successful execution, you can access the app on the url displayed.

6.2 Locally on Machine Using Streamlit

1. Clone the repository:

```
git clone [REPOSITORY-URL]
```

```
cd [REPOSITORY-DIRECTORY]
```

2. Install the required libraries:

```
pip install -r requirements.txt
```

3. Substitute the same Pinecone and Cohere API keys you generated before in 'app.py'.

4. Run the Streamlit app:

```
streamlit run app.py
```

5. Access the app in your browser at **<http://localhost:8501>**.

6.3 Locally Using Docker Desktop

1. Open Docker Desktop.
2. Build the Docker image:

```
docker build -t interactive-qa-bot-app .
```

3. Run the Docker container:

```
docker run -e COHERE_API_KEY=your_cohere_api_key -e  
PINECONE_API_KEY=your_pinecone_api_key -p 8501:8501 interactive-qa-bot-  
app
```

4. Access the app in your browser at **<http://localhost:8501>**.

6.4 Deploying on Streamlit

1. Sign in to your Streamlit account at Streamlit Sharing.
2. Create a new app and link your GitHub repository.
3. Add Secret Keys:

In the Streamlit app settings, navigate to the Secrets section.

Add your API keys securely here which you generated before:

```
PINECONE_API_KEY = "YOUR_PINECONE_API_KEY"
```

```
COHERE_API_KEY = "YOUR_COHERE_API_KEY"
```

4. Follow the prompts to deploy your application.
5. Once deployed, you will receive a live URL for your application, which can be shared with others.