Overview

This RAG-based QA bot leverages a combination of techniques to effectively retrieve and generate relevant information:

1. **Embedding Layer:**
    - **Purpose:** Converts textual data into numerical representations (embeddings) that capture semantic meaning.
    - **Model:** SentenceTransformer ('all-MiniLM-L6-v2') is used to generate high-quality embeddings for both documents and queries.
2. **Vector Database:**
    - **Purpose:** Stores and efficiently retrieves document embeddings based on similarity.
    - **Implementation:** Pinecone is used as the vector database. Documents are indexed using their embeddings, and queries are searched against this index to find the most relevant documents.
3. **Retrieval Mechanism:**
    - **Process:** The query is encoded into an embedding using the same SentenceTransformer model. This embedding is then used to query Pinecone for the nearest neighbor documents.
    - **Similarity Metric:** Cosine similarity is used to measure the similarity between embeddings.
4. **Generative Model:**
    - **Purpose:** Creates coherent text responses based on the retrieved documents and query.
    - **Implementation:** Cohere is used as the generative model. It generates text based on a prompt, which is constructed using the retrieved documents and the query.

Retrieval Process

1. **Embedding Creation:**
    - The query is encoded into an embedding using the SentenceTransformer model.
    - Documents from the dataset are also encoded into embeddings.
2. **Vector Database Search:**
    - The query embedding is searched against the Pinecone index to find the nearest neighbor documents.
    - The top-k most similar documents are retrieved.
3. **Document Retrieval:**
    - The retrieved document IDs are used to fetch the corresponding documents from the original dataset.

Generative Response Creation

1. **Prompt Construction:**
    - A prompt is created for the generative model, combining the query and the retrieved documents.

    - The prompt provides context to the model, guiding it to generate a relevant and informative response.
2. **Text Generation:**
    - The Cohere API is used to generate text based on the prompt.
    - The model leverages its understanding of the language and the context provided in the prompt to produce a coherent and informative response.

Key Components and their Roles:

- **SentenceTransformer:** Encodes text into semantically meaningful embeddings.
- **Pinecone:** Stores and efficiently retrieves embeddings based on similarity.
- **Cohere:** Generates text based on a given prompt.
- **Embedding Layer:** Converts text into numerical representations.
- **Retrieval Mechanism:** Finds the most relevant documents based on query and document embeddings.
- **Generative Response Creation:** Constructs a prompt and uses Cohere to generate a response.

This RAG-based model effectively combines information retrieval and generative capabilities to provide informative and relevant answers to user queries.

EXAMPLE USAGE:

This Streamlit application serves as a question-answering bot that leverages Pinecone and Cohere to process PDFs and provide informative responses.

Prerequisites

- Python 3.11 or later
- Docker (for deployment)
- Pinecone account with API key
- Cohere account with API key
- Streamlit

Installation

1. **Clone the Repository:**
   Bash
   ```
   git clone https://github.com/your-username/your-repo-name.git
   ```

2. **Install Dependencies:**
   Bash
   ```
   cd your-repo-name
   ```

3. ```
   pip install -r requirements.txt
   If you don't have a requirements.txt file, create one with the following content:
   streamlit
   ```
4. ```
   pinecone
   ```
5. ```
   cohere
   ```
6. ```
   sentence-transformers
   ```
7. ```
   PyPDF2
   ```

Usage
   ```

1. **Set API Keys:** Replace the placeholders in the code with your actual Pinecone and Cohere API keys.

2. **Run the Application:**

    - **Locally:** Bash
      ```
      streamlit run app.py
      ```
    - **Using Docker:** Build the Docker image: Bash
      ```
      docker build -t your-app-name .
      ```

      Run the container: Bash
      ```
      docker run -e PINECONE_API_KEY=your_pinecone_api_key
      -e COHERE_API_KEY=your_cohere_api_key -p 8501:8501
      your-app-name
      Access the app at http://localhost:8501/.
      ```

How to Use

1. **Upload PDF:** Click the "Upload a PDF" button and select the PDF file you want to analyze.
2. **Ask Questions:** Enter your question in the text input field and click "Submit."
3. **View Response:** The bot will process the query and provide a relevant answer based on the uploaded PDF.

Configuration

- **Pinecone Index:** The code uses the default index name `"default"`. You can customize this by modifying the `pinecone.Index("default")` line in the `preprocess_and_store_pdf` function.
- **Cohere Model:** Adjust the `model` parameter in the `co.generate` call to use a different Cohere model if desired.

Additional Notes

- Ensure you have a Pinecone index created and your API keys configured correctly.
- For production deployment, consider using a container orchestration platform like Docker Compose or Kubernetes.
- Customize the code to fit your specific requirements, such as adding more features or improving the response generation.