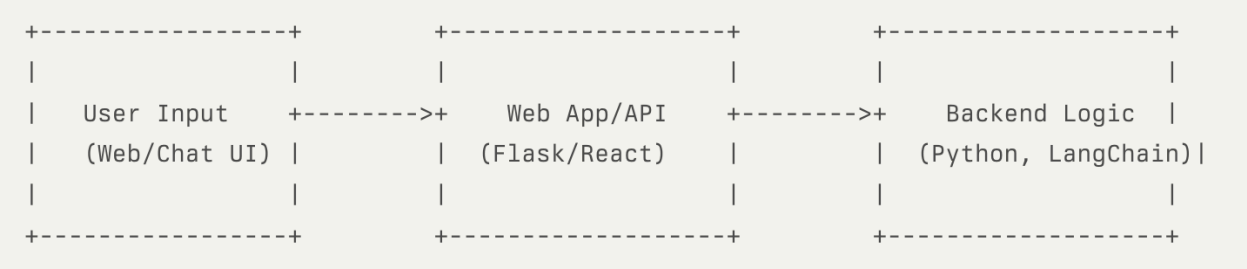
End-to-End Chatbot Project Documentation



Project Overview

This project builds a production-ready medical chatbot using Python, modular coding, vector databases, and large language models (LLMs). The chatbot answers medical questions by retrieving relevant information from a custom knowledge base and generating responses using an LLM. The system is cloud-deployable and features a user-friendly web interface.

1. Project Architecture

- **Knowledge Base:** Extracted from a comprehensive PDF/book.
- **Chunking & Embeddings:** The PDF is split into manageable text chunks, each converted into vector embeddings using a model from Hugging Face.
- **Vector Database:** Embeddings are stored and indexed in Pinecone, a scalable cloud-based vector database.
- **Retrieval:** When a user asks a question, it is embedded and used to search for the most relevant chunks in Pinecone.
- **LLM Response:** Retrieved context and the user's question are sent to an LLM (e.g., OpenAI, Llama, or other supported models) to generate a concise, context-aware answer.
- **Web Interface:** A Flask-based frontend allows users to interact with the chatbot.
- **Version Control & Deployment:** The project uses Git/GitHub for code management and can be deployed to cloud platforms.

2. Tools & Technologies

- **Python 3.10+**
- **LangChain** (for chaining LLM and retrieval logic)
- **Sentence Transformers** (for embeddings)
- **Pinecone** (vector database)
- **Flask** (web framework)
- **Hugging Face** (embedding models)
- **OpenAI or Together AI** (LLMs, can substitute with Llama models)
- **Git/GitHub** (version control)
- **Jupyter Notebook** (for prototyping and experimentation)

3. Step-by-Step Implementation

A. Project Setup

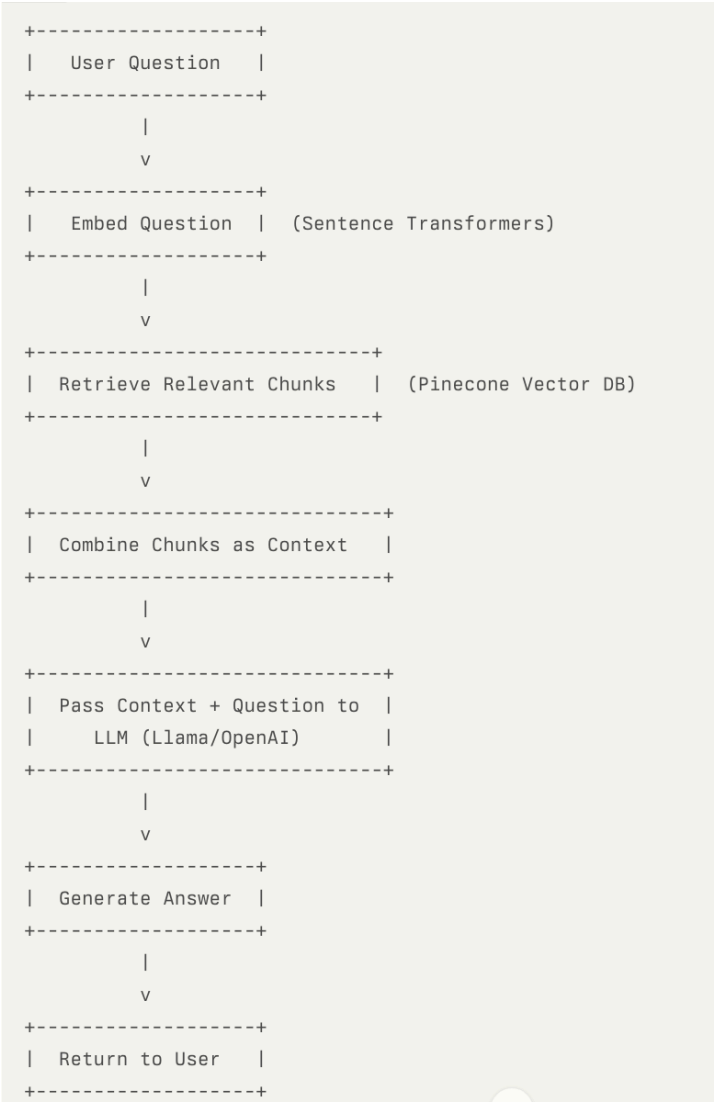
1. **Clone the Repository & Create Virtual Environment**

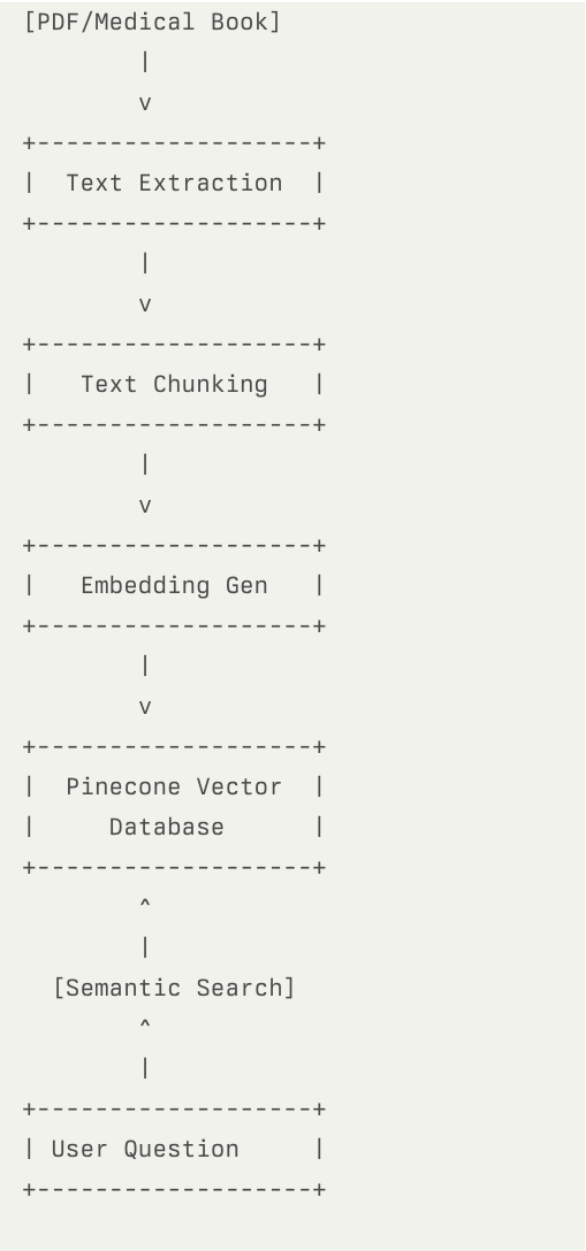
```
bash
git clone <repo_url>
cd <repo_folder>
python -m venv medbot
source bot/bin/activate  # or medbot\Scripts\activate on Windows
```

6. **Install Requirements**

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

8. **Project Folder Structure**





- Use a Python script to automate creation of folders like **src/**, **data/**, **research/**, etc.
- Key files: **app.py**, **src/helper.py**, **src/prompt.py**, **.env**, **requirements.txt**, **setup.py**, **research/trials.ipynb**

B. Data Preparation

1. **Extract Data from PDF**

- Place the medical PDF in the **data/** folder.
- Use **PyPDF** and custom functions to extract text.

2. **Chunk Text**
   - Use LangChain's **RecursiveCharacterTextSplitter** to split the text into chunks (e.g., 500 characters with overlap).
3. **Generate Embeddings**
   - Download a model like **all-MiniLM-L6-v2** from Hugging Face.
   - Convert each chunk to a vector embedding.

C. Vector Database Setup

1. **Create Pinecone Account & Index**
   - Sign up at pinecone.io.
   - Create an index (dimension must match your embedding model, e.g., 384).
2. **Store Embeddings**
   - Use Pinecone's Python SDK to upload chunk embeddings with metadata.

D. Semantic Search & Retrieval

1. **Query Embedding**
   - When a user submits a question, embed it using the same Hugging Face model.
2. **Similarity Search**
   - Query Pinecone to retrieve the most relevant text chunks.

E. LLM Integration & Response Generation

1. **Prepare Prompt**
   - Combine retrieved context and the user's question in a prompt template.
   - Example system prompt:

```
text
You are a helpful medical assistant. Use the following context to answer the question.
If you don't know the answer, say "Sorry, I don't know."
{context}
```

2. **Invoke LLM**
    - Use OpenAI, Together AI, or another supported LLM to generate the answer.

F. Web Interface

1. **Flask App**
    - Build a simple web UI for user input and displaying chatbot responses.
    - Connect Flask endpoints to your backend retrieval and LLM logic.

G. Version Control & Deployment

1. **GitHub**
    - Use Git for version control.
    - Push your project to GitHub for collaboration and backup.
2. **Cloud Deployment**
    - Deploy the app to AWS, GCP, or another cloud provider as needed.

4. Key Code Snippets

Extracting and Chunking PDF:

```
python
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

loader = PyPDFLoader('data/book.pdf')
documents = loader.load()
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=20)
chunks = splitter.split_documents(documents)
```

Generating Embeddings:

```
python
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('all-MiniLM-L6-v2')
embeddings = [model.encode(chunk) for chunk in chunks]
```

Storing in Pinecone:

```
python
import pinecone
```

```
project/
│
├── app.py                # Main Flask app
├── requirements.txt
├── .env
├── data/
│   └── medical_book.pdf
├── src/
│   ├── helper.py         # Helper functions (load, chunk, embed)
│   └── prompt.py         # Prompt templates
├── research/
│   └── trials.ipynb      # Experiments and prototyping
└── ...
```

```
pinecone.init(api_key="YOUR_API_KEY", environment="us-west1-gcp")
index = pinecone.Index("bot")
index.upsert([(str(i), embedding.tolist(), {"text": chunk}) for i, (chunk, embedding) in enumerate(zip(chunks, embeddings))])
```

Semantic Search:

```
python
query_embedding = model.encode(user_query)
results = index.query(query_embedding.tolist(), top_k=5, include_metadata=True)
context = " ".join([match['metadata']['text'] for match in results['matches']])
```

LLM Response:

```
python
prompt = f"You are a helpful assistant. Use the following context to answer the question.\n{context}\nQuestion: {user_query}"
response = llm.generate(prompt)
```

Flask Endpoint Example:

```
python
from flask import Flask, request, jsonify
app = Flask(__name__)

@app.route('/ask', methods=['POST'])
def ask():
    user_query = request.json['query']
    # ...run retrieval and LLM...
    return jsonify({'answer': answer})
```

5. Best Practices

- **Use modular coding:** Separate logic into helper modules and keep code organized.
- **Document your process:** Use README files and code comments for clarity.

- **Version control:** Commit changes regularly and use meaningful commit messages.
- **Test thoroughly:** Try a variety of queries to ensure your chatbot is accurate and reliable.
- **Secure credentials:** Store API keys in **.env** files, never in code.

6. References and Further Learning

- Project Video Walkthrough
- Project Chapters & Materials
- Pinecone Documentation
- LangChain Documentation
- Sentence Transformers
- Flask Documentation

Example: