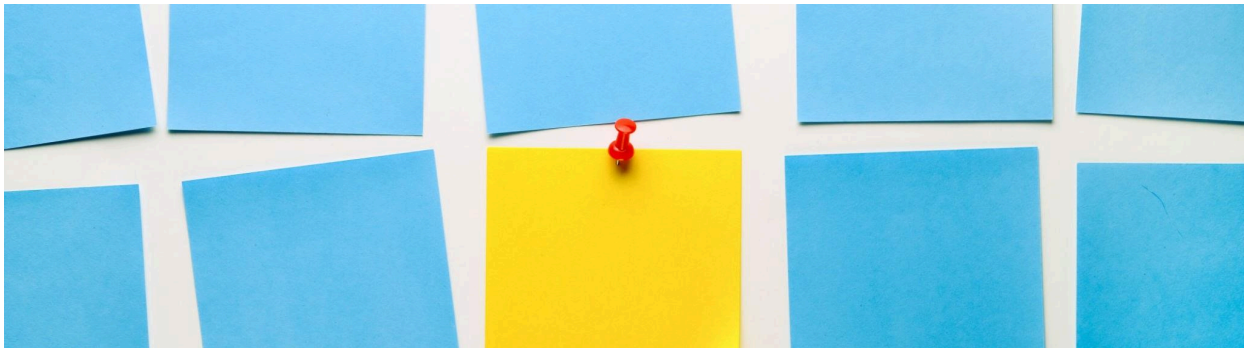


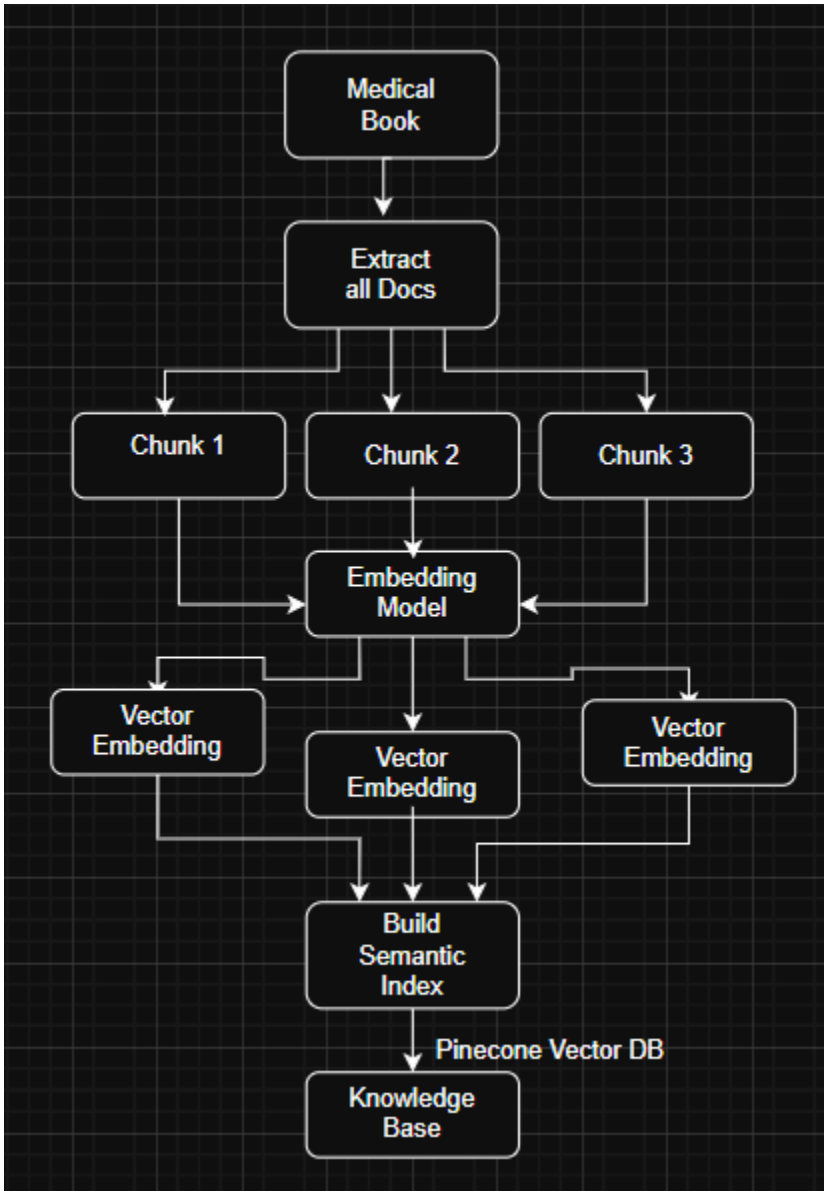
AI CHATBOT

Overview

This project builds an AI-powered document processing pipeline designed to extract, analyze, and retrieve relevant information from large sets of PDF documents. The process begins by loading and extracting text data from multiple PDF files. The extracted text is then split into smaller chunks to make the data easier to handle and process. Each chunk is converted into a numerical vector (embedding) using a pre-trained model from HuggingFace, which transforms the textual information into a format that can be easily compared and searched. These embeddings are then stored in a vector database, Weaviate, which allows for efficient similarity searches and quick retrieval of relevant information based on semantic meaning, rather than exact keyword matching.

The use case for this project is in any scenario where large volumes of text need to be processed and queried efficiently, such as legal document analysis, knowledge management, or research. By using this pipeline, organizations can build powerful search systems where users can query documents using natural language, and the system returns the most relevant text chunks based on context and meaning. This approach is particularly useful for businesses dealing with vast amounts of unstructured data, as it enables them to extract insights quickly and accurately without relying on traditional search methods.

Workflow



This is the Workflow of the Project. OpenAI LLM will be used. We can access it through API requests.

Technology Used

- OpenAI LLM
- Langchain
- Vector Database Weaviate - for API Key
- Flask - To create web application/ UI
- Github - Version Control
- AWS - Deployment

Requirements

- sentence-transformer==2.2.2 → uses Huggingface platform
- Langchain →
- Flask → UI
- pypdf
- Weaviate
- langchain community
- langchani-openai
- Langchain-experimental

Creating File Structure

- **Requirements.txt** → executing this file will download all the libraries which are listed in here.
- **template.py** → this python file is specially created to lessen the effort of manually creating file and folder. You can just simply put any file or folder name, and this python file can execute and create the same for you without duplicating.
- **setup.py** → Setup the Project as Local package

1. Code:

What this code is doing:
This code builds a retrieval-augmented question-answering system that combines document retrieval with a language model (LLM) to answer user questions more intelligently.

API & LLM Setup:

It sets your OpenAI API key and initializes the OpenAI language model with controlled settings (temperature and token limit).

Prompt Design:
A system prompt is defined to guide the assistant's behavior: it tells the LLM to answer based only on the provided context and to admit when it doesn't know the answer. This is a best practice to prevent hallucinations.

Document Chain:
It creates a document chain using create_stuff_documents_chain(). This chain takes the retrieved documents (from your PDF data stored in Weaviate) and injects them into the prompt for the LLM to process when answering questions.

Vector Database Setup:
It connects to your existing Weaviate instance, where you have already stored document embeddings (from your earlier steps). This time, instead of HuggingFace embeddings, you're now using OpenAI's embedding model, which is aligned with your LLM and typically offers stronger performance in OpenAI's ecosystem.

Retriever:
The vector store is turned into a retriever, meaning it can take any user query, convert it into an embedding, and search for the most relevant text chunks (using vector similarity search).

RAG Chain:
Finally, the system builds a Retrieval-Augmented Generation (RAG) chain, which:

First retrieves the most relevant context (text chunks) from Weaviate.

Then uses the LLM to answer the user's question based on this context.

2. Use case:

This setup is powerful for chatbots, knowledge assistants, or customer support tools where you want the system to pull accurate, document-based answers—whether it's from legal files, medical handbooks, company policy PDFs, or research papers.

3. Tech stack:

- Python
- LangChain
- Weaviate
- HuggingFace / OpenAI
- Sentence-transformers & GPT

Summary

This project builds an AI-powered document processing pipeline designed to extract, analyze, and retrieve relevant information from large sets of PDF documents. The process begins by loading and extracting text data from PDF files. The extracted text is then split into smaller chunks to make the data easier to handle and process. Each chunk is converted into a numerical vector (embedding) using a pre-trained model from HuggingFace, which transforms the textual information into a format that can be easily compared and searched. These embeddings are then stored in a vector database, Weaviate, which allows for efficient similarity searches and quick retrieval of relevant information based on semantic meaning, rather than exact keyword matching.

The use case for this project is in any scenario where large volumes of text need to be processed and queried efficiently, such as legal document analysis, knowledge management, or research. By using this pipeline, organizations can build powerful search systems where users can query documents using natural language, and the system returns the most relevant text chunks based on context and meaning. This approach is particularly useful for businesses dealing with vast amounts of unstructured data, as it enables them to extract insights quickly and accurately without relying on traditional search methods.