Gen AI - Intelligent Document Summarization

1. Introduction

This document outlines the basic understanding to read and ask question to retrieve answer from any document for a proof of concept. The study involves to build semantic relationship and summarize text from PDF reports or documents using OpenAI's text-embedded models. One of the most powerful applications enabled by LLMs is sophisticated question-answering (Q&A) chatbots. These are applications that can answer questions about specific source information like PDF, websites, images etc. These applications use a technique known as Retrieval Augmented Generation, or RAG.

The goal is to efficiently extract, process, and summarize content from PDF documents, leveraging advanced natural language processing techniques.

2. Pre-Requisites

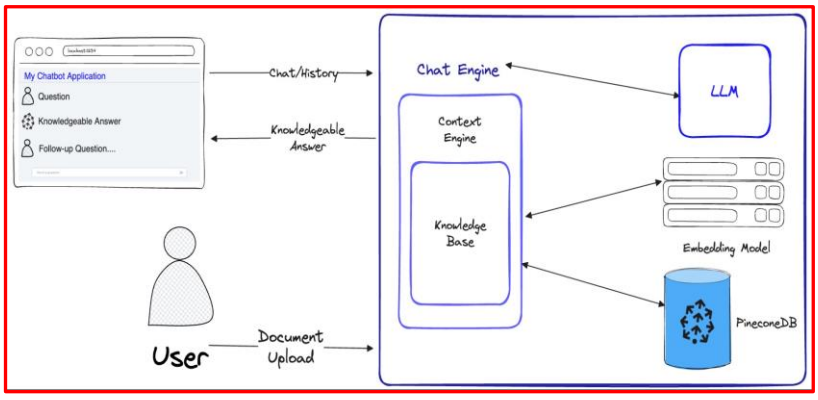
The project requires the installation of several open source packages & dependencies to handle PDF processing, text extraction, and interaction with OpenAI's APIs for text embedding and summarization.

For deployment of application, we can use Streamlit or Amazon services with pinecone DB for storing vectors of word embeddings

Bundle the required packages in a Requirements.txt file

!pip install config
!pip install langchain --upgrade
!pip install pypdf
pip install python-dotenv
!pip install pinecone-client
!pip install openai
!pip install tiktoken
!pip poppler-utils

3. Chat PDF System Workflow

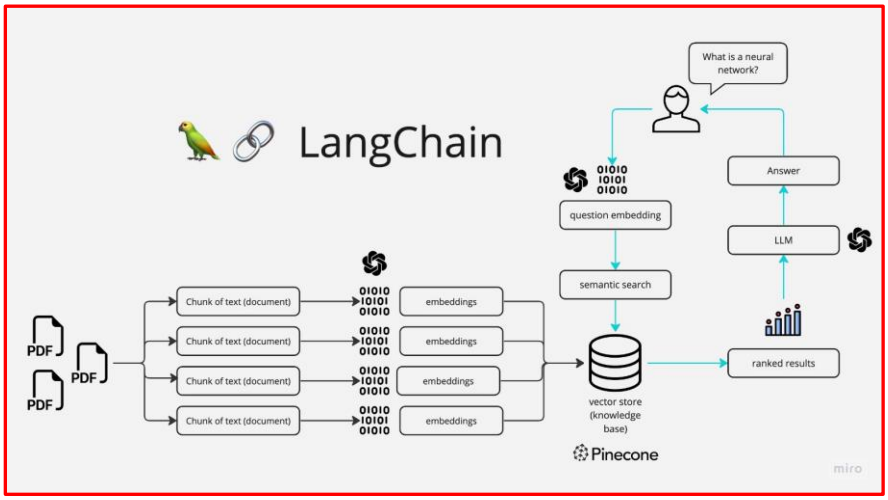


Gen AI - Intelligent Document Summarization

4. Architecture

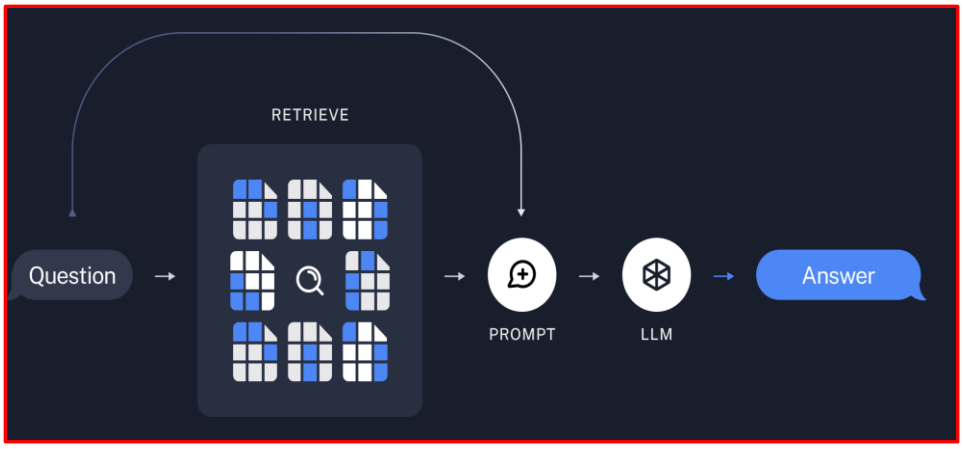
LangChain is a framework designed to simplify the creation of applications using large language models (LLMs). Langchain overlap with language models in general, including document analysis and summarization, chatbots, and code analysis.

RAG is a technique for augmenting LLM knowledge with additional data.



A typical RAG application has two main components:

- Indexing: a pipeline for ingesting data from a source and indexing it. This usually happens offline
- Retrieval and generation: the actual RAG chain, which takes the user query at run time and retrieves the relevant data from the index, then passes that to the model.



Gen AI - Intelligent Document Summarization

3. High Level Process

- Upload and extract text from PDF documents using PyPDF.
- Split extracted text into manageable chunks and prepare it for processing.
- Identify PDF attributes and build metadata for vector database storage.
- Create word embeddings using OpenAI's langchain models.
- Set up and interact with Pinecone's vector database to store and retrieve embeddings.
- Perform similarity searches in the vector database to find relevant text passages.

4. Detailed Level Process

4.1 Text Extraction and Chunking

The process begins with the extraction of text from PDF documents using Document loaders. Extracted text is then split into chunks of 1000 words with an overlap of 200 words.

4.2 Metadata Creation and Storage

Metadata is created for each text chunk to facilitate efficient storage and retrieval. This includes characteristics like character length, page number, and source of a file

4.3 Word Embeddings Generation (Openai , Langchain)

Word embeddings are generated using OpenAI's text-embedding models, which are then used to represent the text chunks in a vector space.

4.4 Vector Database Interaction(PineCone)

Embeddings are stored in Pinecone's vector database. Perform a search on created index and store the sentences /text along with their corresponding similarity matched scores using models.

4.5 Retrieve relevant items from vector DB

Using prompts using a query to bot for relevant text and get the context of same using semantic search as a summarized result.

5. Code Snippets

- Load document into the work environment (goopgle collab/Jupyter notebook)

```
# Load PDF and extract text using PyPDFLoader
from langchain.document_loaders import UnstructuredPDFLoader, OnlinePDFLoader, PyPDFLoader, TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from dotenv import load_dotenv
import os

load_dotenv()
loader = PyPDFLoader("/content/Llama2-Pdf-Chatbot--main/pdfs/GP9_2024.pdf")
```

Gen AI - Intelligent Document Summarization

- Split the document in chunks

```
# We'll split our data into small chunks
from langchain.document_loaders import UnstructuredPDFLoader, OnlinePDFLoader, PyPDFLoader, TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from dotenv import load_dotenv
import os

load_dotenv()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
texts = text_splitter.split_documents(data)
```

```
texts[2]
```

```
Document(page_content='comparable meaning when used in the plural, and vice- versa; (v) words denoting any gender include all genders ; and (vi) \nreferences to (a) articles, exhibits, schedules, attachments, and appendices mean the articles of, and exhibits, schedules, \nattachments, and appendices attached to, this Contract; (b) an agreement, instrument, or other document means such agreement, instrument, or other document as amended, supplemented, and modified from time to time to the extent \npermitted by the provisions thereof; and (c) a statute means such statute as amended from time to time and incl udes any \nsuccessor legislation thereto and any regulations promulgated thereunder. The Parties drafted this Contract without regard \nto any presumption or rule requiring construction or interpretation against the Party drafting an instrument or causing any \ninstrument to be drafted. The exhibits, schedules, attachments, and appendices referred to herein are an integral part of', metadata={'source': '/content/Llama2-Pdf-Chatbot--main/pdfs/GP9_2024.pdf', 'page': 0})
```

- Define the metadata attributes of pdf file(file id , page content etc)

```
text = texts[2]

metadata = {
    'char_length': len(text.page_content),
    'page_no': str(text.metadata["page"]),
    'text': text.page_content[:1000],
    'source': text.metadata["source"]
}
```

```
metadata
```

```
{'char_length': 979,
 'page_no': '0',
 'text': 'acceptance of this Contract in any manner shall conclusively evidence acceptance of this Contract as written. Seller’s \nprovision of Services  shall be governed solely by this Contract . Buyer and Seller are referred to herein as a “Party ” or \ncollectively as the “Parties. ” \nb. Except as authorized herein, no amendment or modification of this Contract shall bind either Party unless it is in writing \nand is signed by the authorized representatives of the Parties.  \nc. For purposes of this Contract, unless the context requires otherwise, (i)  the words “include, ” “includes ,” and “including ” \nare deemed to be followed by the words “ without limitation ”; (ii) the word “ or” is not exclusive; (iii) the words “ herein, ” \n“hereof, ” “hereby, ” “hereto, ” and “hereunder ” refer to this Contract as a whole; (iv) words denoting the singular have a \ncomparable meaning when used in the plural, and vice- versa; (v) words denoting any gender include all genders ; and (vi)',
 'source': '/content/Llama2-Pdf-Chatbot--main/pdfs/GP9_2024.pdf'}
```

- Create Openai embeddings and create vector db index for storage

```
# pip install pinecone-client
from pinecone import Pinecone

pc = Pinecone(api_key="3f93bbd5-79e8-45df-8fdf-36722dffd89b")
index = pc.Index("test")

index.delete(delete_all=True)
```

```
{}
```

```
embeddings = OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY, model="text-embedding-ada-002")
text = texts[1].page_content
response = embeddings.embed_query(text)


response[:1536]
```

Gen AI - Intelligent Document Summarization

```
def createVector(item):

    embeddings = OpenAIEmbeddings(openai_api_key=OPENAI_API_KEY, model="text-embedding-ada-002")
    text = texts[item].page_content[:1000]
    response = embeddings.embed_query(text)
    text = texts[item]


    metadata = {
            'char_length': len(text.page_content),
            'page_no': str(text.metadata["page"]),
            'text': text.page_content[:1000],
            'source': text.metadata["source"]
        }

    vector = []
    vector.append({"id": str(item), "values": response[:1536], "metadata": metadata})
    index.upsert(vector)


for item in range (38,len(texts)):
    print(item)
    createVector(item)
```

- Retrieve answers querying the bot

5. Conclusion

This proof-of-concept study demonstrates the potential of using OpenAI's text-embedded models for summarizing PDF documents. By leveraging advanced NLP techniques and vector database technology, the system can efficiently process and summarize large volumes of text, offering valuable insights from complex documents.

6. References

1. ChatGPT (openai.com)
2. Get started | Langchain
3. Quickstart - Pinecone