Building a PDF chat bot — Retrieval Augmented Generation (RAG)

3 min read · Mar 31, 2024

PrithivirajR Follow

Listen Share

This article will discuss the building of a chatbot using LangChain and OpenAI which can be used to chat with documents. We will discuss the components involved and the functionalities of those components in detail.

Image generated with DALL.E 3

If you haven't had the chance to read my earlier post about Retrieval Augmented Generation (RAG), I recommend checking it out to gain an intuition into how RAG-based systems operate.

Retrieval Augmented Generation (RAG) — An Introduction

This article will provide an overview of prompt engineering and a more in-depth explanation of RAG, aiming to spark…

medium.com

List of components

- Extracting text from PDF documents
- Chunking or Segmentation of text
- Embedding the text and Ingestion in Vectorstore
- Conversation using LLMs (OpenAI)

Extracting text from PDF documents

This component is to extract text from the PDF documents uploaded by the user. The content of the PDF is converted into a raw text format.

```
from PyPDF2 import PdfReader

def get_pdf_content(documents):
    raw_text = ""

    for document in documents:
```

```
    return raw_text
```

The above function extracts text from all the documents provided to it and returns the combined raw text content.

PyPDF2 is a Python library utilized for parsing PDF documents. Using PdfReader(document), you create an instance of a class object with the document, containing page-wise content. This information can then be accessed using pdf_reader.pages(). Within each page, text content can be extracted using the extract_text() method.

Chunking of Combined Text

This component is employed to breakdown the extracted text into chunks, enabling the model to process and generate embeddings more efficiently. Chunking primarily handles the limitations imposed by the model's maximum input token size. This approach can also enhance the model's performance and facilitate the handling of extensive text.

```python
from langchain.text_splitter import CharacterTextSplitter

def get_chunks(text):
    text_splitter = CharacterTextSplitter(
        separator="\n",
        chunk_size=1000,
        chunk_overlap=200,
        length_function=len
    )
    text_chunks = text_splitter.split_text(text)
    return text_chunks
```

The CharacterTextSplitter() module from the langchain library is used specifically for chunking. The parameter 'chunk_size' specifies the token count in each chunk. Additionally, 'chunk_overlap' offers contextual cues from the preceding chunk to the current one, especially if a chunk begins midway through a sentence or word.

Embedding the text and Storage in Vector DB

This module creates embeddings of the text chunks using OpenAI and it is then stored in FAISS (Facebook AI Similarity Search) vector store module. Relevant information can then be retrieved from the vector store.

```
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS

def get_embeddings(chunks):
    embeddings = OpenAIEmbeddings()
    vector_storage = FAISS.from_texts(texts=chunks, embedding=embeddings)

    return vector_storage
```

Conversation using LLMs (OpenAI)

This component initiates a conversation with the specified chat model and sets up memory and adds context by retrieving information from the vector store containing the vector embeddings.

```
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationalRetrievalChain
from langchain.chat_models import ChatOpenAI

def start_conversation(vector_embeddings):
    llm = ChatOpenAI()
    memory = ConversationBufferMemory(
        memory_key='chat_history',
        return_messages=True
    )
    conversation = ConversationalRetrievalChain.from_llm(
        llm=llm,
        retriever=vector_embeddings.as_retriever(),
        memory=memory
    )

    return conversation
```

We'll employ the OpenAI chat model to generate a response for the provided user query.

Putting it all together

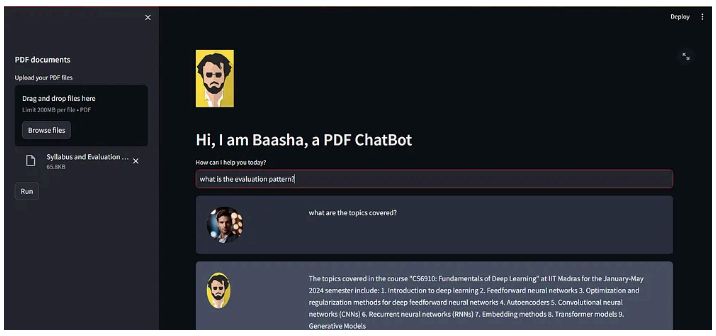
The complete implementation of the project with the HTML template has been provided in my github repo. To run the project, add your OpenAI API key in the .env file.

GitHub - Prithiviraj7R/Chat-with-PDF

Contribute to Prithiviraj7R/Chat-with-PDF development by creating an account on GitHub.

github.com

Results



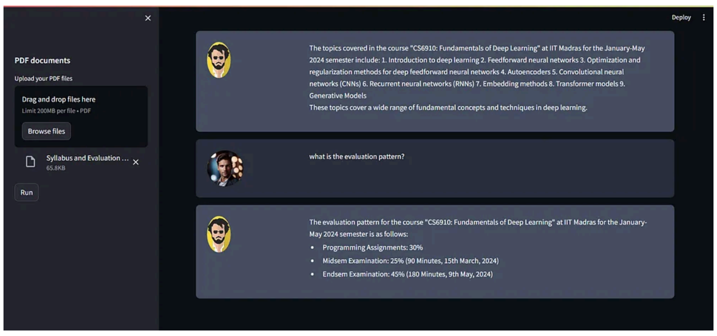
Hi, I am Baasha, a PDF ChatBot

How can I help you today?

what is the evaluation pattern?

what are the topics covered?

The topics covered in the course "CS6910: Fundamentals of Deep Learning" at IIT Madras for the January-May 2024 semester include: 1. Introduction to deep learning 2. Feedforward neural networks 3. Optimization and regularization methods for deep feedforward neural networks 4. Autoencoders 5. Convolutional neural networks (CNNs) 6. Recurrent neural networks (RNNs) 7. Embedding methods 8. Transformer models 9. Generative Models



The topics covered in the course "CS6910: Fundamentals of Deep Learning" at IIT Madras for the January-May 2024 semester include: 1. Introduction to deep learning 2. Feedforward neural networks 3. Optimization and regularization methods for deep feedforward neural networks 4. Autoencoders 5. Convolutional neural networks (CNNs) 6. Recurrent neural networks (RNNs) 7. Embedding methods 8. Transformer models 9. Generative Models
These topics cover a wide range of fundamental concepts and techniques in deep learning.

what is the evaluation pattern?

The evaluation pattern for the course "CS6910: Fundamentals of Deep Learning" at IIT Madras for the January-May 2024 semester is as follows:

- Programming Assignments: 30%
- Midsem Examination: 25% (90 Minutes, 15th March, 2024)
- Endsem Examination: 45% (180 Minutes, 9th May, 2024)

Demonstration of the project

References

Introduction | 🦜️🔗 Langchain

LangChain is a framework for developing applications powered by language models. It enables applications that:

python.langchain.com