Chat with PDFs | Empowering Textual Interaction with Python and OpenAI

ADVANCED CHATBOT DATABASE GENERATIVE AI LLMS PYTHON

Introduction

In a world filled with information, PDF documents have become a staple for sharing and preserving valuable data. However, extracting insights from PDFs hasn't always been straightforward. That's where "Chat with PDFs" comes to the rescue – an innovative project revolutionising how we interact with PDFs.

In this article, we introduce you to the fascinating "Chat with PDFs" project, which combines the power of Language Model Libraries (LLMs) and the versatility of PyPDF's Python library. This unique fusion allows you to have natural conversations with your PDF documents, making it easy to ask questions and get contextually relevant answers.

Learning Objectives

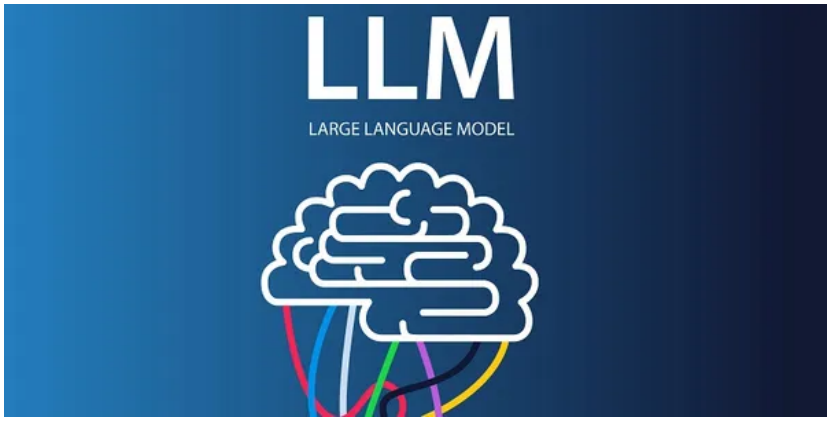
- Gain insight into Language Model Libraries (LLMs) as advanced AI models capable of understanding human language patterns and generating meaningful responses.
- Explore PyPDF, a versatile Python library, to comprehend its functionalities for text extraction, merging, and splitting in PDF manipulation.
- Recognize the integration of Language Model Libraries (LLMs) and PyPDFs in creating an interactive chatbot for natural conversations with PDFs.

This article was published as a part of the Data Science Blogathon.

Table of contents

Understanding Language Model Libraries (LLMs)

The heart of “Chat with PDFs” lies in Language Model Libraries (LLMs), advanced AI models trained on vast amounts of text data. Think of them as language experts, capable of understanding human language patterns and generating meaningful responses.



For our project, LLMs play a vital role in creating an interactive chatbot. This chatbot can process your questions and understand what you need from the PDFs. The chatbot can provide helpful answers and insights by tapping into the knowledge base hidden in PDFs.

PyPDFs – Your PDF Super Assistant

PyPDF is a versatile Python library that simplifies interactions with PDF files. It equips users with various functionalities, such as text extraction, merging, and splitting of PDF documents. This library is a vital component of our project, as it enables seamless handling of PDFs and streamlines the subsequent analysis.

PyPDF helps us load PDF files and extract their text within our project, setting the stage for efficient processing and analysis. With this powerful assistant, you can interact with PDFs effortlessly.

Chat with PDFs liberates PDF documents from their static state by bringing together Language Model Libraries (LLMs) and PyPDFs. Now, you can explore your PDFs like never before, extracting valuable information and engaging in meaningful conversations. From academic papers to business reports, “Chat with PDFs” makes interacting with PDFs a delightful experience.

So, let’s dive into the fascinating world of Chat with PDFs project.

Project

```
# Importing necessary libraries and setting up API keys import os import pandas as pd import matplotlib.pyplot as plt from transformers import GPT2TokenizerFast from langchain.document_loaders import PyPDFLoader from langchain.text_splitter import RecursiveCharacterTextSplitter from langchain.embeddings import OpenAIEmbeddings from langchain.vectorstores import FAISS from langchain.chains.question_answering import load_qa_chain from langchain.llms import OpenAI from langchain.chains import ConversationalRetrievalChain from key import openaiapi_key os.environ["OPENAI_API_KEY"] = openaiapi_key
```

The code above initiates the “Chat with PDFs” project by importing essential libraries and setting up the API keys. We use the ‘os’ library to interact with the operating system, ‘pandas’ for data manipulation, and ‘matplotlib’ for plotting graphs. The ‘transformers’ library provides the ‘GPT2TokenizerFast’ class, which is essential for tokenizing text. The ‘langchain’ modules include classes necessary for loading PDFs (‘PyPDFLoader’), text splitting (‘RecursiveCharacterTextSplitter’), embeddings (‘OpenAIEmbeddings’), vector storage (‘FAISS’), question-answering chains (‘load_qa_chain’), language models (‘OpenAI’), and conversational chains (‘ConversationalRetrievalChain’).

PyPDF Loader

We then use the ‘PyPDFLoader’ class to load the PDF file and split it into separate pages. Finally, we print the first page’s content to verify the PDF’s successful loading and splitting.

```
# Simple method - Split by pages loader = PyPDFLoader("story.pdf") # We're creating an instance of 'PyPDFLoader' and passing the file path of #the PDF we want to work with. pages = loader.load_and_split() print(pages[0])
```

This section covers the loading and chunking of the PDF document. Two methods are demonstrated: the simple method that splits the PDF by pages and the advanced method that involves converting the PDF to text and splitting it into smaller chunks.

```
# Advanced method - Split by chunk # Step 1: Convert PDF to text import textract doc = textract.process("story.pdf") # Step 2: Save to .txt and reopen (helps prevent issues) with open('story.txt', 'w') as f: f.write(doc.decode('utf-8')) with open('story.txt', 'r') as f: text = f.read() # Step 3: Create function to count tokens tokenizer = GPT2TokenizerFast.from_pretrained("gpt2") def count_tokens(text: str) -> int: return len(tokenizer.encode(text)) # Step 4: Split text into chunks text_splitter = RecursiveCharacterTextSplitter( # Set a really small chunk size, just to show. chunk_size = 512, chunk_overlap = 24, length_function = count_tokens, ) chunks = text_splitter.create_documents([text])
```

Steps for Splitting the PDF

The advanced method splits the PDF into smaller chunks for more efficient processing. We achieve this through the following steps:

Step 1: We use the ‘textract’ library to extract text from the PDF file and store it in the ‘doc’ variable.

Step 2: We save the extracted text to a text file (‘story.txt’) to prevent potential issues and reopen it in read mode. The content is stored in the ‘text’ variable.

Step 3: We define a function called ‘count_tokens’ to count the number of tokens in a given text. This function uses the ‘GPT2TokenizerFast’ class to tokenize the text.

Step 4: Using the ‘RecursiveCharacterTextSplitter’ class, we split the ‘text’ into smaller ‘chunks’ to ensure efficient processing, with each chunk having a maximum token limit.

```
# Embed text and store embeddings # Get embedding model embeddings = OpenAIEmbeddings() # Create vector database db = FAISS.from_documents(chunks, embeddings)
```

OpenAI Embeddings

In this section, we embed the text using the ‘OpenAIEmbeddings’ class, which converts text into numerical representations (embeddings). These embeddings facilitate efficient storage and analysis of textual data. We then create a vector database using the ‘FAISS’ class, incorporating the ‘chunks’ of text and their corresponding embeddings.

```
# Setup retrieval function # Check similarity search is working query = "What is the name of the author?" docs = db.similarity_search(query) docs[0]
```

Set up a retrieval function in this part. You can perform a similarity search with a sample query using the vector database (‘db’). The variable query contains the question we want to ask the chatbot and the variable docs store the relevant documents containing the query’s context. We then print the first document returned from the similarity search.

```
chain = load_qa_chain(OpenAI(temperature=0), chain_type="stuff") query = "What is the name of the author?" docs = db.similarity_search(query) chain.run(input_documents=docs, question=query)
```

In this segment, we create a question-answering chain (‘chain’) that integrates the similarity search with user queries. We load the ‘OpenAI’ language model and set the temperature to 0 for deterministic responses. We obtain an answer based on the knowledge base by passing the retrieved documents (‘docs’) and the user’s question (‘query’) to the chain.

```
# Create chatbot with chat memory from IPython.display import display import ipywidgets as widgets qa = ConversationalRetrievalChain.from_llm(OpenAI(temperature=0.1), db.as_retriever()) chat_history = [] def on_submit(_): query = input_box.value input_box.value = "" if query.lower() == 'exit': print("Thank you for using the Chat with PDFs chatbot!") return result = qa({"question": query, "chat_history": chat_history}) chat_history.append((query, result['answer'])) display(widgets.HTML(f'<b>User:</b> {query}')) display(widgets.HTML(f'<b><font color="blue">Chatbot:</font></b>{result["answer"]}')) print("Welcome to the Chat with PDFs chatbot! Type 'exit' to stop.") input_box = widgets.Text(placeholder='Please enter your question:') input_box.on_submit(on_submit) display(input_box)
```

In the final section, we introduce a chatbot feature where Users can interact with the chatbot by entering questions and get answers.

Conclusion

This article explored the fascinating “Chat with PDFs” project and its step-by-step implementation. We’ve gained a deeper understanding of Language Model Libraries (LLMs) and PyPDFs, two essential components powering this innovative tool. Now you can effortlessly process and analyze PDF documents, extracting valuable insights and engaging in interactive conversations with a chatbot companion. Whether you’re a researcher, student, or professional, “Chat with PDFs” has revolutionized how we interact with PDFs, making the previously static documents come to life with the power of AI. Happy PDF exploring!

Key Takeaways

1. LLMs empower the chatbot to deliver accurate and context-aware responses to user queries.
2. PyPDF simplifies PDF manipulation, making it easier to work with complex documents.
3. The code’s structure ensures smooth integration of text embedding and similarity search functionalities.
4. PyPDF enables seamless PDF handling, text extraction, and manipulation.

Frequently Asked Questions

Q1. What are LLMs, and how do they work?

A. Large Language Models are powerful AI models trained on vast amounts of text data. They can understand human language and perform various natural language processing tasks, such as text generation, summarization, and question-answering.

Q2. What is the role of LLMs?

A. LLMs create a chatbot that can answer user queries based on information extracted from PDF documents. The LLM processes and understands user questions and provides relevant responses from the knowledge base.

Q3. What is PyPDF? How does it help with PDF handling?

A. PyPDF is a Python library designed for working with PDF files. It enables tasks like text extraction, merging, and splitting of PDF documents, making it a valuable tool for PDF-related tasks in a programmatic manner.

Q4. How is PyPDF used in PDFs?

A. PyPDF loads pdf files and extracts their content into pdfs. The extracted text is then processed and split into smaller chunks, facilitating efficient processing and analysis.

Article Url - https://www.analyticsvidhya.com/blog/2023/08/chat-with-pdfs/

Gayathri Jujjuru