PDF reader documentation

Llm model used is: orca-mini-3b.ggmlv3.q4\_0.bin

The orca-mini-3b.ggmlv3.q4\_0.bin model is a variant of the Orca Mini 3B model trained by Pankaj Mathur and it's based on the OpenLLaMa-3B model. It has been trained on "explain tuned" datasets, which were created using instructions and input from the WizardLM, Alpaca, and Dolly-V2 datasets.

# Importing Libraries:

The code starts by importing several Python libraries for different tasks, including reading PDFs, text processing, language modeling, and progress bar visualization. Notable libraries include PyPDF2, pdfplumber, SentenceTransformer, GPT4All, and numba.

# Progress Bar Function:

progressBar is a custom function used to display a progress bar in the console. It takes parameters like count\_value, total, and suffix to show the progress of a task.

# Loading the GPT-4 Model:

The code initializes a GPT-4 language model using the "orca-mini-3b.ggmlv3.q4\_0.bin" model file with the GPT4All library.

# User Input Function:

get\_query() function prompts the user to enter a question and then increments the progress bar.

# PDF Text Extraction:

load\_split\_pdf(pdf\_path) function reads a PDF file (pdf\_path) using PdfReader and extracts the text from all pages using pdfplumber. The extracted text is then concatenated. This is the second step of the progress bar.

# Text Splitting:

split\_text\_using\_RCTS(pdf\_text) function splits the extracted PDF text into paragraphs using a custom text splitter called RecursiveCharacterTextSplitter. The split paragraphs are stored in a list. This is the third step of the progress bar.

Sentence Embedding Model Initialization:

Initialize\_sentence\_transformer() function initializes a sentence embedding model called "sentence-transformers/all-MiniLM-L6-v2" using the SentenceTransformer library. Sentence embeddings are used to compare query and paragraph similarity. This is the fourth step of the progress bar.

# Encoding Each Paragraph:

encode\_each\_paragraph(paragraphs, embeddings) function encodes each paragraph using the initialized sentence embedding model. The encoded paragraphs are stored along with their original text. This is the fifth step of the progress bar.

# Choosing Relevant Sentences:

choose\_most\_relevant\_sentence(embeddings, responses, query) function computes the similarity between the user's query and each paragraph using cosine similarity. If the similarity is above a threshold (0.6), the paragraph is considered relevant, and the relevant paragraphs are concatenated into an "answer." This is the sixth step of the progress bar.

# Querying the GPT-4 Model:

query\_the\_llm(answer, llm\_model, query) function combines the user's query and the relevant paragraphs to form a prompt. It then uses the GPT-4 model (llm\_model) to generate a response. The final response is stored in final\_response.

# Main Function:

The main() function orchestrates all the previously explained steps in sequence. It also calculates the execution time and displays the final response along with the execution time. Finally, it completes the progress bar.

# Execution:

The script's execution is triggered by calling the main(llm) function at the end, passing the GPT-4 model as an argument.

In summary, this code takes a user's question, processes a PDF document, extracts relevant paragraphs, computes sentence embeddings, identifies relevant paragraphs based on query similarity, and generates a final response using the GPT-4 language model. It provides a comprehensive pipeline for extracting information from a PDF document and generating responses to user queries.