Explanation of the system:

First step is to take a document and divide it into smaller pieces. It is an important process because when we're recalling it and querying it in order to get an answer based on the document, we need to receive a bunch of smaller chunks that is relevant to user query. Not the whole information. Here each chunk is 512 tokens or less. The next step is to embed each chunk one by one. So, here I've used ADA-002 model by openai. Then all of these embedding for each chunk are put into a vector database. So, they are ready for recall upon user queriers. Then the final step is to allow users to query the database. It is done by taking the user queries, putting through the exact same embedding model. That's how it returns a number of documents most similar to user queries. Then the similar documents are passed to the LLM. Here both user queries and the matched documents are passed to the LLM model to generate a response.

The whole process is conducted in Google Colaboratory.

The code begins by importing various libraries required for different functionalities, such as file handling (os), data manipulation (pandas), data visualization (matplotlib.pyplot), natural language processing (transformers.GPT2TokenizerFast), and the necessary components from the langchain library for document loading, text splitting, embeddings, vector stores, question answering, and conversational retrieval

The OpenAI API key is set using the os.environ command, which allows the code to access the OpenAI API.

The PDF document ("Redmon_You_Only_Look_CVPR_2016_paper.pdf") is converted to text using the textract library. The extracted text is saved to a file named "YOLO.txt".

The previously saved text file ("YOLO.txt") is opened and read into the text variable.

A function named count_tokens is defined to count the number of tokens in a given text. The function uses the GPT-2 tokenizer (GPT2TokenizerFast) to encode the text and returns the count of tokens.

The text is split into smaller chunks using the RecursiveCharacterTextSplitter from the langchain library. The splitter divides the text into chunks based on a specified chunk size, while considering the token count obtained from the count_tokens function. The resulting chunks are stored in the chunks variable.

An embedding model is created using the OpenAIEmbeddings class from the langchain library. This model will be used to generate embeddings for the text chunks.

A vector database is created using the FAISS class from the langchain library. The chunks of text obtained from the previous step are added to the database along with their corresponding embeddings generated by the embedding model.

The code imports the necessary components for creating a chatbot interface, such as display from IPython.display and widgets from ipywidgets.

A conversational retrieval chain is created using the ConversationalRetrievalChain class from the langchain library. This chain is initialized with the OpenAI language model (OpenAI) with a temperature of 0.1, and the vector database is used as the retriever in the chain. This allows the chatbot to generate responses based on the input question and chat history.

The code defines an on_submit function that is triggered when the user enters a question and submits it. The function retrieves the user's query, clears the input box, and checks if the query is "exit" to stop the chatbot. If not, the chatbot generates a response using the conversation chain's qa method, which takes the user's query and chat history as input. The question and answer are then appended to the chat history. The user's query and the chatbot's answer are displayed using HTML formatting.

The code displays a welcome message and creates an input box for the user to enter their questions. The on_submit function is connected to the input box, so when the user submits a question, it triggers the function to generate a response.

Challenges and Limitations:

One of the main challenges that I've faced during the implementation was the lack of availability of Openai api key. On a free account the api key that openai provides has a certain credit limit. It means only a certain number of tokens can be processed with it. That's why I had to use a small pdf as the knowledge base of the chatbot. If I could use a large pdf then I think the chatbot would have been more lucrative.

The chatbot has a conversational memory but it's not too powerful. Sometimes it doesn't grab the whole user query but it still manages to response accordingly. And as it was done in Google Colab, there's no UI for this chatbot.

How to Run:

As it is created in Google Colab, anyone can run by going into https://colab.research.google.com/drive/1okDsEFJhCXnxtBb3t7_3il9kADVraIuC?usp=sharing this link. First you have to upload a pdf and change the name of the pdf in code. And copy and paste your own openai api key in 4th cell. After this, just run all the cells and you're ready to go.

Demonstration:

The demonstration video can be found via this link: https://drive.google.com/file/d/1NYoqNiRKmxfi9_ABBb74aOIzySNbjWMm/view?usp=sharing