**AI Document Retrieval Chatbot**

I recently worked on an AI project focused on developing a chatbot designed to efficiently retrieve and present information from company-related PDF documents. The primary objective was to build a tool that could accurately answer user queries based on data stored in these PDFs, streamlining access to important company details and improving internal operations.

**Problem Statement:**

The goal was to create a simple yet effective chatbot capable of answering questions related to company information. The challenge was that all the relevant data was stored in PDF format, which required careful extraction and processing to be usable by the chatbot.

**Data Extraction and Preprocessing:**

We began by extracting text from the company's PDF documents. For this task, we utilized tools like **PyPDF**, **PDFPlumber**, and **PDFMiner**, which are specifically designed to handle the complexities of PDF text extraction. These tools allowed us to access and extract the necessary data from the PDFs, converting it into a raw text format.

Once the text was extracted, the next step was to preprocess the data. We applied Natural Language Processing (NLP) techniques to clean and organize the data. . The steps include converting text to lowercase, removing stop words, lemmatization, parts of speech tagging, special character removal, URL removal, emoticon conversion, converting smileys to actual emojis, and emoji removal. This steps use for interfere with the chatbot's ability to understand and retrieve accurate information. The preprocessing phase was crucial in ensuring that the data was clear and well-structured for the next steps.

**Data Chunking and Embedding:**

After preprocessing, we needed to break down the text into smaller, more manageable pieces, a process known as chunking. We used **LangChain** to divide the text into coherent chunks, ensuring that each chunk contained relevant and easily retrievable information.

These chunks were then converted into numerical embeddings using word embedding techniques. This conversion was necessary for the chatbot to understand and process the text data in a meaningful way. The embeddings were stored in **Chroma DB**, a vector database chosen for its efficiency in handling and retrieving large volumes of vectorized data.

**Model Integration and Retrieval:**

To retrieve information and generate responses, we employed a **Retrieval-Augmented Generation (RAG)** framework. This approach combines retrieval techniques with generative models to provide accurate and contextually relevant answers to user queries. For the generative component, we used OpenAI's **GPT-3.5 Turbo** model, known for its advanced natural language processing capabilities.

When a user interacts with the chatbot, their query triggers a semantic similarity search within the Chroma DB. This search identifies the most relevant chunks of information based on the user's input. The RAG framework then uses this information to generate a precise and informative response, effectively answering the user's query.

We developed a Flask API that serves as the backend for the chatbot. Whenever a user interacts with the chatbot, their input is processed through a semantic similarity search, matching their query with the most relevant information stored in the vector database. The GPT-3.5 Turbo model then generates a response based on this information .