**Project Overview: AI Based Chatbot with Semantic Search and Citations**

Project Description

The goal of this project is to create an AI>based chatbot that can answer user queries using information from a set of PDF documents. The chatbot uses semantic search to find relevant information, combines retrieval with generative response using Retrieval>Augmented Generation (RAG), and provides citations for the information it retrieves.

Key Features

1. Semantic Search:

> Instead of relying solely on keyword matching, the chatbot understands the meaning behind the queries, allowing for more accurate and context>aware responses.

2. Document Chunking:

> The PDF documents are divided into fixed>size chunks to make processing easier and ensure that the chatbot can effectively retrieve relevant sections.

3. RAG (Retrieval>Augmented Generation):

> This approach combines retrieved documents with the generative capabilities of models like GPT>2, allowing the chatbot to formulate coherent and contextually relevant responses.

4. Citations:

> The chatbot provides references to the original documents from which it retrieves information, ensuring transparency and credibility.

5. Session Management:

> Each conversation can be titled, and the chatbot maintains a chat history for review and logging.

Technology Stack

> Python: The primary programming language used.

> Transformers Library: Used for the GPT>2 model and tokenizer.

> Sentence Transformers: Used to create document embeddings for semantic search.

> FAISS: A library for efficient similarity search and clustering of dense vectors.

> Streamlit: A framework for building web applications easily, which serves as the user interface.

Code Breakdown

1. chatbot.py

> Functionality:

> Loads text chunks and initializes the GPT>2 model.

> Handles user input, retrieves relevant documents, and generates responses.

2. faiss\_store.py

> Functionality:

> Creates a FAISS index from document embeddings.

> Implements functions to retrieve documents based on user queries.

3. utils.py

> Functionality:

> Contains helper functions, including logging chat history to a file.

4. app.py

> Functionality:

> Sets up the Streamlit app, allowing users to input their questions and receive responses from the chatbot.

How It Works

1. User Input: The user types a question in the Streamlit interface.

2. Document Retrieval: The chatbot processes the input, retrieves relevant document chunks using semantic search.

3. Response Generation: The chatbot generates a response using the retrieved information, providing citations where applicable.

4. Chat History: The interaction is logged for future reference.

Challenges and Solutions

> Understanding Natural Language: Initially, the chatbot struggled with context. Using sentence embeddings helped improve its understanding.

> Response Coherence: Generating coherent responses required fine>tuning the generation parameters of the GPT>2 model.

> Citations: Ensuring the chatbot provided accurate citations required careful tracking of the source of each response.

Conclusion

This project showcases how AI and NLP can be combined to create intelligent conversational agents capable of providing reliable and contextually relevant information. The use of semantic search and RAG techniques enhances the quality of responses, making the chatbot a valuable tool for users seeking information.