## Overview

This software implements a chatbot interface using **Streamlit**, **OpenAI's GPT model**, and **HuggingFace embeddings**. The goal is to assist users with questions about the content of **CS 410: Text Information Systems**, providing contextually relevant answers based on stored lecture content. It utilizes a **retrieval-augmented generation (RAG)** approach for accurate and contextual responses.

---

## Key Components

### 1. Dependencies

- **os**: Used to manage environment variables and file operations.
- **shutil**: For copying and renaming files during text retrieval and parsing.
- **pandas**: For handling CSV data and creating DataFrames.
- **streamlit**: For creating a user-friendly web interface.
- **langchain_openai**: Provides the interface to OpenAI's GPT model.
- **langchain_huggingface**: Used for embedding generation using a HuggingFace model.
- **langchain_community.vectorstores**: Manages the FAISS vector store for document similarity search.
- **langchain_core.prompts**: Handles prompt templates.
- **langchain_core.output_parsers**: Parses output from the GPT model.

### 2. Environment Setup

- **OPENAI_API_KEY**: Set as an environment variable to authenticate with the OpenAI API.

### 3. Data

- **cs-410**: Source directory containing raw text files.
- **scraped_files**: Destination directory where parsed text files are stored.
- **content_vectors.csv**: A CSV file containing lecture content and its precomputed embeddings.
    - Columns:
        - `Concatenated`: The textual data for each document.
        - `vectors`: The precomputed embeddings as Python lists.
    - This is loaded into a Pandas DataFrame for processing.

### 4. Embedding Model

- **HuggingFace Embeddings**:

    - Model: `Alibaba-NLP/gte-large-en-v1.5`.
    - Device: `cpu`.
    - Trusts remote code execution.
    - Configured for non-normalized embeddings and batch processing.

**5. Vector Store Creation**

- **Function: `vector_store_faiss`**:
    - Takes a DataFrame, embeddings, an embedding model, and metadata parameters.
    - Creates a FAISS vector store for efficient similarity search.
    - Parameters:
        - `df`: DataFrame with content and metadata.
        - `embeddings`: Precomputed embeddings.
        - `embeddings_model`: Model used for embeddings.
        - `pc_col` and `metadata_cols`: Columns for text and metadata, respectively.

---

## Implementation Details

**1. Text Retrieval and Parsing**

- **Initial File Scraping**:
    - Source directory: `cs-410`.
    - Destination directory: `scraped_files`.
    - Function: `scrape_en_txt_files`:
        - Walks through the source directory to find `.en.txt` files.
        - Copies the files into the destination directory, preserving subfolder structure.
        - Ensures destination subfolders are created if they don't exist.
        - Preserves file timestamps and metadata.
- **File Renaming**:
    - Function: `rename_files`:
        - Renames files in the destination directory to include their subfolder name as a prefix.
        - Ensures unique and descriptive file names.

**2. Vector Store Creation**

The lecture content is used to generate a vector store:

- Precomputed vectors are loaded from `content_vectors.csv`.

- A FAISS vector store is created using the text and embedding pairs.

### 3. Streamlit User Interface

- **Title and Description**:
  - Displays the title and an introductory description of the chatbot.
- **Session State**:
  - Maintains user interaction history in `st.session_state.messages`.
  - Appends user messages and chatbot responses.

### 4. Chat Interaction Workflow

- **User Input**:
  - Captured via `st.chat_input()`.
- **Document Retrieval**:
  - The user's question is processed with `similarity_search` to retrieve the top 5 relevant documents.
  - Each document is represented with metadata (`Week`, `Lesson`) and its content.
- **Prompt Construction**:
  - Combines retrieved documents into a formatted context.
  - Constructs a detailed prompt for the GPT model, ensuring it adheres to the context provided.
- **Response Generation**:
  - The prompt is passed to the GPT model using `llm.invoke()`.
  - The response is displayed to the user via `st.chat_message()` and stored in session state.
- **Caching**
  - Caches components like the embeddings model, vector store, QA chain, and prompt template.
  - Responses are cached for one hour to improve efficiency.