Documentation for `Legal Document Assistant`

Overview

The **Legal Document Assistant** is a Python-based tool designed to process, summarize, and query legal documents. Built using state-of-the-art NLP libraries and Hugging Face's transformers, it allows users to upload legal documents, filter them based on metadata (e.g., jurisdiction and date), and ask questions about their content. The system uses FAISS for efficient document retrieval and Gradio for a user-friendly interface.

Features

- **Upload and Process Documents**: Supports both `.pdf` and `.txt` file formats for legal documents.
- **Metadata Handling**: Allows tagging documents with metadata such as jurisdiction and date.
- **Summarization**: Uses the Hugging Face model `facebook/bart-large-cnn` for generating concise summaries of legal content.
- **Vector Search**: Implements FAISS for storing and retrieving document chunks based on semantic similarity.
- **Querying with Filters**: Allows querying with jurisdiction and date filters to narrow down relevant legal documents.
- **Interactive Chat Interface**: Gradio-powered UI to upload documents, ask legal questions, and view summaries.

File Structure and Documentation

1. Setup and Dependencies

The notebook requires the following dependencies:

- **LangChain**: For document processing, embedding, and vector search.
- **Hugging Face Transformers**: For summarization and embeddings.
- **FAISS**: For vector-based document retrieval.
- **Gradio**: For building the web-based interactive user interface.
- **PyPDF2**: For extracting text from PDF files.
- **spaCy**: For additional text-processing utilities.

2. Global Variables

- `vectorstore`: Stores the FAISS index for document embeddings.
- `stored_documents`: Stores processed document chunks for filtering and searching.

3. Utility Functions

`process_file(file)`

- **Purpose**: Reads and extracts text from `.pdf` or `.txt` files.
- **Input**:
    - `file`: An uploaded file object.
- **Output**: Extracted text as a string.
- **Logic**:
    - Uses `PyPDF2` for PDF parsing.
    - Decodes text files directly.

`preprocess_documents(file_text, metadata=None)`

- **Purpose**: Splits large text files into manageable chunks with metadata.
- **Input**:
    - `file_text`: Full text of the document.
    - `metadata`: Dictionary containing metadata (jurisdiction, date).
- **Output**: List of document chunks.
- **Logic**: Uses `RecursiveCharacterTextSplitter` for chunking.

`summarize_text_with_hugging_face(text, model_name="facebook/bart-large-cnn")`

- **Purpose**: Summarizes a given text using Hugging Face's `facebook/bart-large-cnn`.
- **Input**:
    - `text`: Text to summarize.
    - `model_name`: Hugging Face model for summarization.
- **Output**: Summarized text.
- **Logic**: Uses `pipeline` from Hugging Face transformers.

`create_vectorstore(documents)`

- **Purpose**: Creates or updates a FAISS vectorstore for semantic search.

- **Input**:
    - `documents`: List of processed document chunks.
- **Output**: FAISS vectorstore object.
- **Logic**: Uses `HuggingFaceEmbeddings` to convert text to embeddings.

---

4. Gradio App Functions

`upload_and_process(file, jurisdiction, date)`

- **Purpose**: Handles file upload, processes the document, and stores metadata.
- **Input**:
    - `file`: Uploaded legal document file.
    - `jurisdiction`: Jurisdiction metadata.
    - `date`: Date metadata.
- **Output**: Success message.
- **Logic**: Combines file processing, text preprocessing, and vectorstore creation.

`ask_question_with_filters(query, jurisdiction, date, chat_history=[])`

- **Purpose**: Answers questions about the document using filters and summarization.
- **Input**:
    1. `query`: User's legal question.
    2. `jurisdiction`: Filter for jurisdiction.
    3. `date`: Filter for date.
    4. `chat_history`: Chatbot history for context.
- **Output**: Updated chat history.
- **Logic**:
    1. Filters stored documents based on metadata.
    2. Creates a temporary FAISS index for filtered documents.
    3. Retrieves the most relevant documents.
    4. Summarizes the retrieved content using Hugging Face.

---

5. Gradio UI

The user interface consists of:

1. **Upload Section**:
    - File upload.
    - Metadata inputs (jurisdiction and date).
    - Upload button to trigger processing.

2. **Chat Section**:
    - Chatbot to display Q&A history.
    - Input fields for query, jurisdiction, and date filters.
    - Button to ask questions.

How to Run the Application

1. **Install Dependencies**: Ensure all required packages are installed using the provided `pip install` commands.
2. **Launch the App**: Run the notebook, and the Gradio app will start at a local URL.
3. **Upload Documents**:
    - Click "Upload Legal Document" and choose a `.pdf` or `.txt` file.
    - Optionally, add jurisdiction and date metadata.
    - Click "Upload and Process."
4. **Ask Questions**:
    - Enter your legal question in the query input box.
    - Optionally, filter by jurisdiction or date.
    - Click "Ask" to get summarized answers.

Example Usage

1. **Upload Document**:
    - File: `eurojust_guidelines.pdf`
    - Jurisdiction: `EU`
    - Date: `2022`
2. **Ask a Question**:
    - Query: "What are the guidelines for Eurojust?"
    - Jurisdiction: `EU`
    - Date: `2022`

Output: A summarized answer extracted from the most relevant document.

Key Libraries and Tools

1. **LangChain**:
    - For document chunking, embedding, and vector retrieval.
2. **Hugging Face**:
    - For text summarization (`facebook/bart-large-cnn`) and embeddings.
3. **FAISS**:

    - For efficient similarity search.
4. **Gradio**:
    - For building an interactive user interface.
5. **PyPDF2**:
    - For PDF text extraction.

Limitations and Future Enhancements

Limitations:

- Summarization length is capped to prevent excessively long outputs.
- Large document processing may be resource-intensive.
- Limited jurisdiction/date filters.

Enhancements:

1. Add support for additional summarization models.
2. Extend metadata tagging for more complex filters.
3. Integrate more sophisticated document parsing libraries for better text extraction.
4. Enable persistent storage of vectorstores to avoid reprocessing.

Conclusion

The **Legal Document Assistant** simplifies working with legal texts by enabling efficient document upload, metadata filtering, and question answering. This tool is ideal for legal professionals looking for quick insights from large collections of legal documents.