LEGAL DOCUMENT PROCESSING – **DocChat**

**Introduction**

This project is an **AI-powered Document Summarization and Chatbot System** designed to automate the extraction, cleaning, and summarization of text from various document formats. Built using state-of-the-art machine learning models, it provides a robust solution for interacting with uploaded documents through a user-friendly chatbot interface.

The system's core functionality revolves around converting lengthy documents into concise summaries, extracting key information, and allowing users to interact with this data conversationally. The chatbot can handle general conversations as well as document-specific inquiries, making it versatile for both casual and professional use cases.

The primary objectives of this project are:

1. **Automated Document Processing**: Streamline the extraction of text from various formats such as PDF, DOCX, and images (through OCR).
2. **Text Summarization**: Use fine-tuned models like bart-large-cnn or t5-small to generate meaningful summaries of the extracted content.
3. **Chatbot Integration**: Offer a dynamic chat interface where users can upload documents, receive summaries, and ask questions based on the document’s content.
4. **File Generation**: Allow users to request PDF or DOCX files containing summaries or chat transcripts for future reference.
5. **Storage and Retrieval**: Store chat history and document details in a database for easy access and retrieval.

**Key Features:**

* **Text Extraction**: Handles documents in multiple formats, including PDFs, DOCX files, and images.
* **Data Cleaning**: Cleans the extracted text to ensure noise is removed, improving the quality of summaries.
* **Summarization**: Generates structured summaries using state-of-the-art machine learning models.
* **Chat Interface**: Provides an intuitive chatbot interface powered by models like Qwen110B, enabling users to ask questions about the document.
* **File Export**: Automatically generates PDF or DOCX files based on chat conversations and document summaries.
* **History Tracking**: Saves chat history and document information in an SQLite database for future reference.

This documentation outlines the entire project, detailing each module and file, along with step-by-step instructions on how to set up, run, and extend the system. Whether you're using the chatbot to gain insights from contracts, reports, or research papers, this system provides an interactive and efficient way to process large volumes of text.

**Project Structure**

This section provides an overview of the project’s file structure and the purpose of each file. The system consists of various Python scripts, data files, and model configurations that work together to accomplish the desired document extraction, summarization, and chatbot functionality.

Below is a breakdown of the project files and directories:

**1. .venv/**

* **Description**: The virtual environment directory containing all the dependencies required for running the project.

**2. .vscode/**

* **Description**: Configuration files for the Visual Studio Code IDE, tailored to the project for easier debugging and code navigation.

**3. Sample\_Contracts/**

* **Description**: A folder containing sample documents (such as contracts or research papers) used for testing the document extraction and summarization processes.

**4. \_\_pycache\_\_/**

* **Description**: A directory generated by Python, storing compiled bytecode files to enhance the performance of the application.

**5. .env**

* **Description**: Environment variables file, containing sensitive information like API keys, tokens, and other project-specific configurations. For security, API keys (like the Hugging Face token) are stored here.

**6. bot\_testing\_phase1.py**

* **Description**: A testing script that was used in the initial stages of the chatbot development. It served to validate the bot's conversational capabilities during the initial project phase.

**7. chatbot\_history.db**

* **Description**: An SQLite database that stores the chat history, allowing the system to keep track of previous conversations and document interactions. This can be used to retrieve past conversations related to uploaded files.

**8. chatbot\_streamlit.py**

* **Description**: The primary script for running the chatbot with a **Streamlit** interface. It enables document upload, text extraction, summarization, and interaction through the chat UI.

**9. chathistorywen110b.txt**

* **Description**: A text file storing raw chat history generated during the use of the Qwen110BChat.py script for debugging or review purposes.

**10. cleaning.py**

* **Description**: A core script responsible for cleaning the extracted text. It removes unnecessary characters, symbols, and formats the text for better summarization and processing.

**11. database.py**

* **Description**: Manages the connection to the SQLite database (chatbot\_history.db) and includes functions to save and retrieve chat history based on the uploaded documents and user interactions.

**12. generate\_file.py**

* **Description**: A utility script responsible for generating PDF and DOCX files from chat history or user-specific requests. It creates files on demand containing summaries or chat transcripts.

**13. keyWord\_Extraction.py**

* **Description**: Extracts key information such as phone numbers, emails, company names, addresses, and other relevant details from the document. This helps in creating more structured summaries and responding to user queries more effectively.

**14. Qwen110BChat.py**

* **Description**: The chatbot script powered by the Qwen110B model. It handles user interactions, responds to document-related queries, and generates conversational replies based on the model.

**15. requirements.txt**

* **Description**: A file listing all the Python dependencies required to run the project. The necessary packages can be installed using pip based on this file.

**16. st-Qwen1.5-110B-Chat.py**

* **Description**: Another chatbot script that builds upon Qwen110BChat.py, enhancing the interaction capabilities of the chatbot model. This version also includes some UI adjustments and user prompts specific to this model.

**17. summarize.py**

* **Description**: The summarization module that interacts with the document extraction pipeline. This script generates summaries using models such as t5-small or bart-large-cnn. It can also identify key details to create structured summaries.

**18. Text\_Extraction.py**

* **Description**: A script responsible for extracting text from **PDF** and **DOCX** files. It serves as the entry point for document processing before cleaning and summarization.

**19. Text\_Extraction\_img.py**

* **Description**: A supplementary text extraction script that handles the extraction of text from images using Optical Character Recognition (OCR). It works in tandem with Text\_Extraction.py for documents in image formats.

**20. user\_response\_bot.py**

* **Description**: This script manages user interaction, guiding the conversation flow, and integrating with the document processing system to provide meaningful responses based on the user queries.

**Additional Files/Notes:**

* **Directory-specific Hidden Files**: Files like .dist and .env handle deployment and environment settings, while project-specific cache files (such as \_\_pycache\_\_) improve performance.

This detailed file structure provides a high-level overview of how different components interact in the project. Each module plays a distinct role in ensuring that text is extracted, cleaned, summarized, and presented in an intuitive chatbot format.

**Setup and Installation**

This section outlines the necessary steps to set up and run the project on your local machine or deployment server. Follow the instructions below to ensure a smooth installation and execution process.

**1. Prerequisites**

Before setting up the project, ensure you have the following software and tools installed:

* **Python 3.8 or later**: You can download Python from [python.org](https://www.python.org/).
* **Virtual Environment Tool**: (Recommended) venv or virtualenv to manage project dependencies in an isolated environment.
* **Git**: Version control tool for cloning the repository.
* **pip**: Python package manager, typically bundled with Python.

**2. Cloning the Project**

Start by cloning the project repository. Open a terminal window and run:

bash

Copy code

git clone

<https://github.com/EntertainmentTechnologists/sushain_CheckLegalDocumentProcessing>

Replace <https://github.com/EntertainmentTechnologists/sushain_CheckLegalDocumentProcessing>

with the actual URL of the GitHub repository or source location.

**3. Setting Up the Virtual Environment**

Once the project is cloned, it’s recommended to create a virtual environment to manage dependencies.

bash

Copy code

cd <<https://github.com/EntertainmentTechnologists/sushain_CheckLegalDocumentProcessing>>

python3 -m venv .venv

After creating the virtual environment, activate it using the following commands:

* On **Windows**:

bash

Copy code

.\.venv\Scripts\activate

* On **MacOS/Linux**:

bash

Copy code

source .venv/bin/activate

**4. Installing Dependencies**

With the virtual environment activated, install the required dependencies listed in the requirements.txt file.

bash

Copy code

pip install -r requirements.txt

This will ensure that all the necessary Python libraries, including streamlit, PyPDF2, transformers, and others, are installed correctly.

**5. Setting Up Environment Variables**

The project uses environment variables for secure and flexible configurations, such as API keys and access tokens.

1. Open the .env file located in the project root directory.
2. Replace placeholders with the actual values (e.g., Hugging Face token, API keys). Example .env file format:

bash

Copy code

HF\_Token=<Your\_Hugging\_Face\_Token>

API\_KEY=<Your\_API\_Key>

Save the .env file after updating the values.

**6. Running the Application**

The core application is powered by **Streamlit**, which serves the chatbot interface. To run the chatbot, follow these steps:

1. Open the terminal.
2. Make sure the virtual environment is activated.
3. Run the Streamlit app:

bash

Copy code

st-Qwen1.5–110B-Chat.py

This command will launch the Streamlit interface, and you can access it in your browser (typically at http://localhost:8501/ by default).

**7. Database Setup (Optional)**

The project uses SQLite for storing chatbot history (chatbot\_history.db). No additional setup is required, as the database is automatically created and managed by the project scripts.

However, if needed, you can inspect or modify the database using SQLite tools.

**8. Testing and Validation**

After the setup, you can test the application by:

* Uploading documents (e.g., PDFs or DOCX files) to check the text extraction and summarization process.
* Interacting with the chatbot for document-related queries and responses.
* Requesting the generation of files (PDFs or DOCX) based on user prompts.

**9. Deactivating the Virtual Environment**

Once you're done working on the project, you can deactivate the virtual environment by running:

bash

Copy code

deactivate

This ensures that your global Python environment remains unaffected by project-specific dependencies.

**Project Structure**

The project is organized into several files and directories, each with a specific purpose to ensure modularity and maintainability. Below is a detailed explanation of the folder and file structure:

**Root Directory Structure**

markdown

Copy code

> \_pycache\_

> .dist

> .venv

> .vscode

> Sample\_Contracts

✡ .env

bot\_testing phase1.py

chatbot\_history.db

chatbot\_streamlit.py

chathistorywen110b.txt

cleaning.py

database.py

generate\_file.py

keyWord\_Extraction.py

Qwen110BChat.py

requirements.txt

st-Qwen1.5-110B-Chat.py

summarize.py

Text\_Extraction\_img.py

Text\_Extraction.py

user\_response\_bot.py

**1. *pycache* Directory**

* **Purpose**: Contains compiled Python bytecode files generated automatically by Python for performance optimization. This folder is auto-generated and does not require modification.

**2. .dist Directory**

* **Purpose**: This directory may be used for storing distribution-related files such as package builds. Typically used when preparing for deployment or distribution of the project.

**3. .venv Directory**

* **Purpose**: Contains the virtual environment for the project, including all project-specific dependencies and Python packages installed via pip.
* **Note**: This folder should not be modified manually and is essential for keeping project dependencies isolated.

**4. .vscode Directory**

* **Purpose**: Holds configuration files for Visual Studio Code, such as workspace settings, which are useful if using VS Code as the primary IDE.

**5. Sample\_Contracts Directory**

* **Purpose**: This folder contains sample contracts used for testing the chatbot and text extraction processes. These files are used to validate the accuracy of the extraction, cleaning, and summarization steps.

**6. .env File**

* **Purpose**: This file contains environment variables such as API keys and access tokens required for interacting with external services (e.g., Hugging Face, Gemini API).
* **Key Variables**:
  + HF\_Token: Hugging Face API token.
  + API\_KEY: Gemini API key.

**7. bot\_testing phase1.py**

* **Purpose**: A testing script used during the initial phases of chatbot development. It might contain test cases or scenarios to validate the chatbot’s basic functionalities.

**8. chatbot\_history.db**

* **Purpose**: SQLite database file where the chatbot’s conversation history is stored. Each conversation is linked to an uploaded document and saved for future retrieval.

**9. chatbot\_streamlit.py**

* **Purpose**: This is the main application script that runs the chatbot interface via Streamlit. It handles the user interface, document uploads, and interaction logic between the user and chatbot.
* **Key Functions**:
  + **File Upload**: Allows users to upload PDF and DOCX files.
  + **Chat Interface**: Displays the chatbot interaction, including chat history and document-based responses.
  + **Response Generation**: Manages the integration between document text, user queries, and chatbot responses.

**10. chathistorywen110b.txt**

* **Purpose**: A plain text file used for testing or storing chat history. This file could be part of development experiments using the Qwen110B model or other chatbot experiments.

**11. cleaning.py**

* **Purpose**: This script is responsible for cleaning and preprocessing the extracted text. It ensures that the text is free from noise, irregular characters, and other artifacts that might affect the summarization process.
* **Key Function**:
  + **clean\_text**: Cleans the extracted text, removing unwanted characters and formatting issues.

**12. database.py**

* **Purpose**: Handles all database-related operations for saving and retrieving chatbot conversation history. The chatbot\_history.db file is managed through this script.
* **Key Functions**:
  + **Save Chat**: Inserts new conversations and responses into the SQLite database.
  + **Retrieve Chat**: Fetches previous conversation history based on document uploads.

**13. generate\_file.py**

* **Purpose**: Generates files (PDF or DOCX) based on user queries or chatbot responses. This is particularly useful when users request downloadable files summarizing a document or specific content from the chat history.
* **Key Functions**:
  + **Generate PDF**: Creates a PDF file from a given text input.
  + **Generate DOCX**: Creates a DOCX file from a given text input.

**14. keyWord\_Extraction.py**

* **Purpose**: Extracts key information such as phone numbers, company names, addresses, and email addresses from the uploaded documents. This script aids in structuring and organizing the document summaries more effectively.

**15. Qwen110BChat.py**

* **Purpose**: A script for chatbot experiments using the Qwen110B model. This model was tested for quality and response coherence during chatbot development. It could also include logic for handling specific types of user queries or document interactions.

**16. requirements.txt**

* **Purpose**: A text file listing all the Python dependencies required to run the project. Installing these dependencies ensures the project has all the necessary libraries and packages.
* **Key Libraries**:
  + streamlit: Used for the chatbot UI.
  + transformers: For handling pre-trained models and text generation.
  + PyPDF2: For extracting text from PDF documents.
  + sqlite3: For database management.

**17. st-Qwen1.5-110B-Chat.py**

* **Purpose**: A Streamlit version of the chatbot code using the Qwen1.5-110B model for interaction. This file likely mirrors the functionality in chatbot\_streamlit.py but uses a different model for text generation and response.

**18. summarize.py**

* **Purpose**: Handles the summarization of the extracted and cleaned text. It uses a pre-trained or fine-tuned model (such as T5-small or BART) to generate concise summaries of long documents.
* **Key Function**:
  + **Summarize**: Takes the cleaned text and generates a structured summary, which can be used by the chatbot for responding to user queries.

**19. Text\_Extraction\_img.py**

* **Purpose**: Responsible for extracting text from image files (e.g., JPG, PNG) using Optical Character Recognition (OCR). This script supports documents where text needs to be extracted from images, not just PDF or DOCX formats.

**20. Text\_Extraction.py**

* **Purpose**: Extracts text from documents (e.g., PDF, DOCX). This is the core of the text extraction process for uploaded files. It ensures that the text is ready for cleaning and summarization.
* **Key Libraries**:
  + PyPDF2: For handling PDF documents.
  + docx: For handling DOCX files.

**21. user\_response\_bot.py**

* **Purpose**: This script contains the logic for generating chatbot responses based on user input and extracted document content. It might include integration with the fine-tuned T5 model or other response generation models to ensure the chatbot replies appropriately.

**Project Workflow**

The project follows a streamlined workflow for document processing and chatbot interaction, ensuring efficient handling of user-uploaded documents and generating structured responses. Below is a step-by-step description of the workflow:

**1. Document Upload**

* **Process**: Users can upload documents in PDF, DOCX, or image formats via the Streamlit interface.
* **Script**: chatbot\_streamlit.py
* **Action**: The chatbot interface prompts the user to upload a file, which is then processed in the subsequent steps.

**2. Text Extraction**

* **Process**: Once the document is uploaded, the relevant text is extracted depending on the file type (PDF, DOCX, or image).
* **Scripts**:
  + Text\_Extraction.py (for PDFs and DOCX files)
  + Text\_Extraction\_img.py (for image files)
* **Action**: Extracted text is passed through these scripts to convert the document into clean, readable text for further processing.

**3. Text Cleaning**

* **Process**: After text extraction, the raw text often contains unwanted characters, formatting issues, or noise. The text is cleaned using a predefined cleaning function.
* **Script**: cleaning.py
* **Action**: The clean\_text function removes any unwanted elements, ensuring the text is ready for summarization and further processing.

**4. Key Information Extraction**

* **Process**: In this step, important information like phone numbers, email addresses, company names, and addresses are identified and extracted from the document text.
* **Script**: keyWord\_Extraction.py
* **Action**: The script parses the text to find and extract key information that will be integrated into the final structured output.

**5. Text Summarization**

* **Process**: The cleaned text, along with the extracted key information, is summarized to provide a concise version of the document.
* **Script**: summarize.py
* **Model**: Uses fine-tuned models such as T5-small or BART to generate structured summaries.
* **Action**: Summarized text is generated based on the cleaned document content. This summary includes the extracted key information, providing a comprehensive overview of the document.

**6. Chatbot Interaction**

* **Process**: Once the document is summarized, the chatbot generates responses based on the document's content and user queries.
* **Scripts**:
  + user\_response\_bot.py
  + Qwen110BChat.py
  + st-Qwen1.5-110B-Chat.py
* **Action**: The chatbot answers user queries with the help of document-based content, ensuring that responses are accurate and relevant to the uploaded file.

**7. Conversation History Storage**

* **Process**: Every chatbot interaction, along with the document and user input, is saved to a database for future reference.
* **Script**: database.py
* **Action**: The conversation history is stored in chatbot\_history.db. Each interaction is associated with the uploaded document name, allowing users to retrieve chat history based on specific documents.

**8. File Generation**

* **Process**: Users may request to download chatbot responses or document summaries in either PDF or DOCX format.
* **Script**: generate\_file.py
* **Action**: This script generates downloadable files based on the chatbot's output, allowing users to export summarized content.

**9. Display and Customization**

* **Process**: The chatbot interface is visually appealing, with a dark theme, custom styling, and a typewriter effect when displaying success messages.
* **Script**: chatbot\_streamlit.py
* **Action**: The user experience is enhanced through UI customization, ensuring the chatbot feels intuitive and professional.

**Implementation Details**

This section outlines the technical details, tools, and libraries used to implement each stage of the project.

**1. Document Upload**

* **Tool**: Streamlit (Frontend Framework)
* **Script**: Part of the UI design for document upload functionality.
* **Details**:
  + The file upload feature allows users to upload PDF, DOCX, or image files.
  + Upon successful upload, the file is processed to extract text using the respective extraction scripts.

**2. Text Extraction**

* **Tools**:
  + PyPDF2 (for PDF extraction)
  + python-docx (for DOCX extraction)
  + Tesseract OCR (for image text extraction)
* **Scripts**:
  + Text\_Extraction.py: Extracts text from PDFs and DOCX files.
  + Text\_Extraction\_img.py: Uses Tesseract OCR to extract text from images.
* **Details**:
  + **PDFs**: Extracted using the extract\_text\_from\_pdf function.
  + **DOCX**: Extracted using the extract\_text\_from\_docx function.
  + **Images**: Extracted using extract\_text\_from\_img function powered by Tesseract OCR.

**3. Text Cleaning**

* **Script**: cleaning.py
* **Details**:
  + **Functionality**: Removes unnecessary characters, corrects punctuation, and refines the text for easier processing.
  + The clean\_text function processes text extracted from documents to eliminate any irrelevant content.
  + The cleaning step ensures that the text is ready for further stages like classification or chatbot interactions.

**4. Chat History and Database Integration**

* **Script**: database.py
* **Details**:
  + The system stores chat history for each document session in an SQLite database.
  + Each conversation is associated with an uploaded file, allowing users to revisit previous interactions based on the document name.
  + The database stores the chat history using the following format:
    - [FILE\_NAME]
    - [CHAT\_HISTORY]

**5. Large Language Model Integration**

* **Script**: Qwen110BChat.py, st-Qwen1.5-110B-Chat.py
* **Details**:
  + These scripts integrate with the Qwen 110B large language model (LLM) to enable the chatbot to generate contextually relevant responses.
  + The system uses a hidden prompt feature to ensure that responses are tailored to the uploaded documents, even when multiple files are used in a single conversation.
  + Integration with Hugging Face API enables communication with pre-trained models for effective language understanding.

**6. File Generation and Export**

* **Script**: generate\_file.py
* **Details**:
  + The chatbot allows users to generate PDFs or DOCX files based on chat history or requested outputs.
  + Users can choose the file format (PDF or DOCX), and the system will compile the chat conversation or relevant document summaries into the selected format.

This covers the core functionalities of the chatbot system, focusing on document handling, text extraction, processing, and chat history management.