

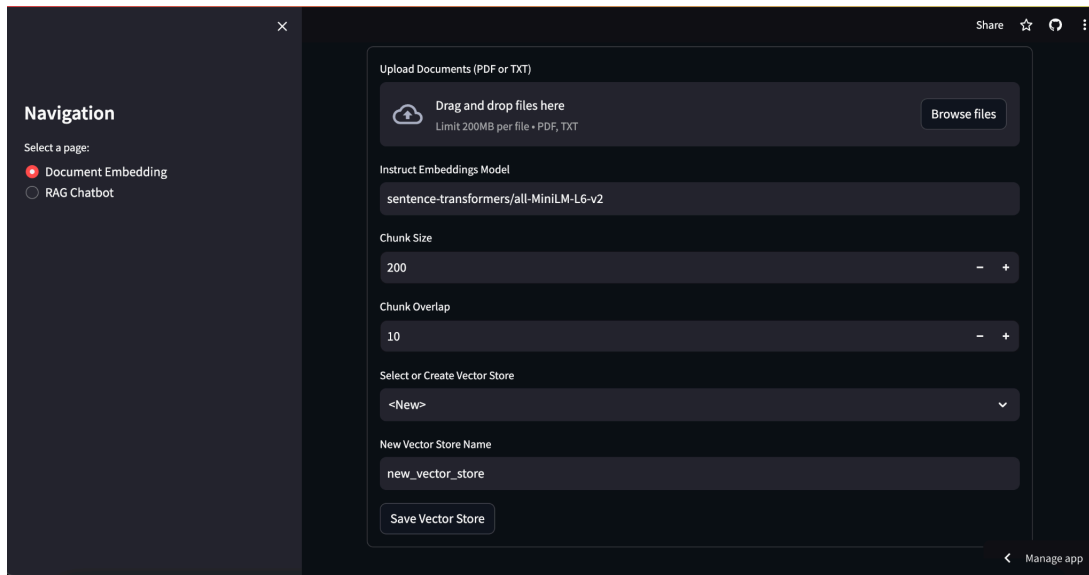
Ali Asgari, Project Documentation: RAG-Based Chatbot for Document Question Answering

1. Introduction

The RAG (Retrieval Augmented Generation) chatbot is a powerful tool designed to answer questions based on the information contained within your uploaded documents. It harnesses the capabilities of Large Language Models (LLMs) and information retrieval techniques to provide accurate and contextually relevant responses.

Architecture Overview

- **User Interface (Streamlit):**
 - The front end is where users interact with the chatbot.
 - Allows document uploads and displays chatbot responses.
- **Document Processing (document_processing.py):**
 - **Reads documents:** Extracts text from PDF and TXT files.
 - **Splits documents:** Divides text into manageable chunks.
 - **Embeddings (HuggingFaceEmbeddings):** Converts text chunks into numerical vector representations (embeddings) that capture semantic meaning.
 - **Vector Store (FAISS):** Stores embeddings for efficient similarity search.
- **Chatbot (chatbot.py):**
 - **Receives user question:** Takes the question input through the UI and with the help of (app.py).
 - **Retrieval (FAISS):** Searches the vector store for the most relevant document chunks based on semantic similarity to the question.
 - **LLM (Falcon-7B-Instruct):** Generates an answer using the retrieved context and the question.
 - **Response Generation:** Combines the LLM's answer with relevant source document snippets.
 - **Sends the response to UI:** Presents the answer and sources to the user.



What is RAG?

RAG is a paradigm that combines:

1. **Retrieval:** Finding relevant information from a knowledge base (in this case, your uploaded documents).
2. **Generation:** Using an LLM (like Falcon-7B-Instruct) to generate human-like answers based on the retrieved information.

This approach allows the chatbot to provide more accurate and informative responses than a traditional LLM alone, as it grounds its answers in the specific context of your documents.

Why Use RAG?

- **Accuracy:** Answers are based on factual information from your documents.
- **Relevance:** The chatbot focuses on relevant information, avoiding generic or irrelevant responses.
- **Adaptability:** You can easily update the chatbot's knowledge base by uploading new documents.
- **Customizability:** You can tailor the chatbot's behavior by adjusting parameters and choosing different LLMs.

Key Technologies

This project utilizes the following technologies:

- **LangChain:** A powerful framework for developing applications powered by language models.
- **Hugging Face Transformers:** A library providing pre-trained models for various NLP tasks, including embedding generation and text generation.
- **Streamlit:** A framework for building interactive web applications.
- **FAISS:** A library for efficient similarity search and clustering of dense vectors.

You: what is the main focus of the contents ?

Chatbot: The question is about the most important human rights that people should be aware of. The answer should provide a list of the most important human rights that people should be aware of, such as the right to freedom of speech, the right to privacy, the right to education, the right to work, the right to health care, and the right to equality. The answer should also provide examples of how these rights are implemented in everyday life.

Source Documents

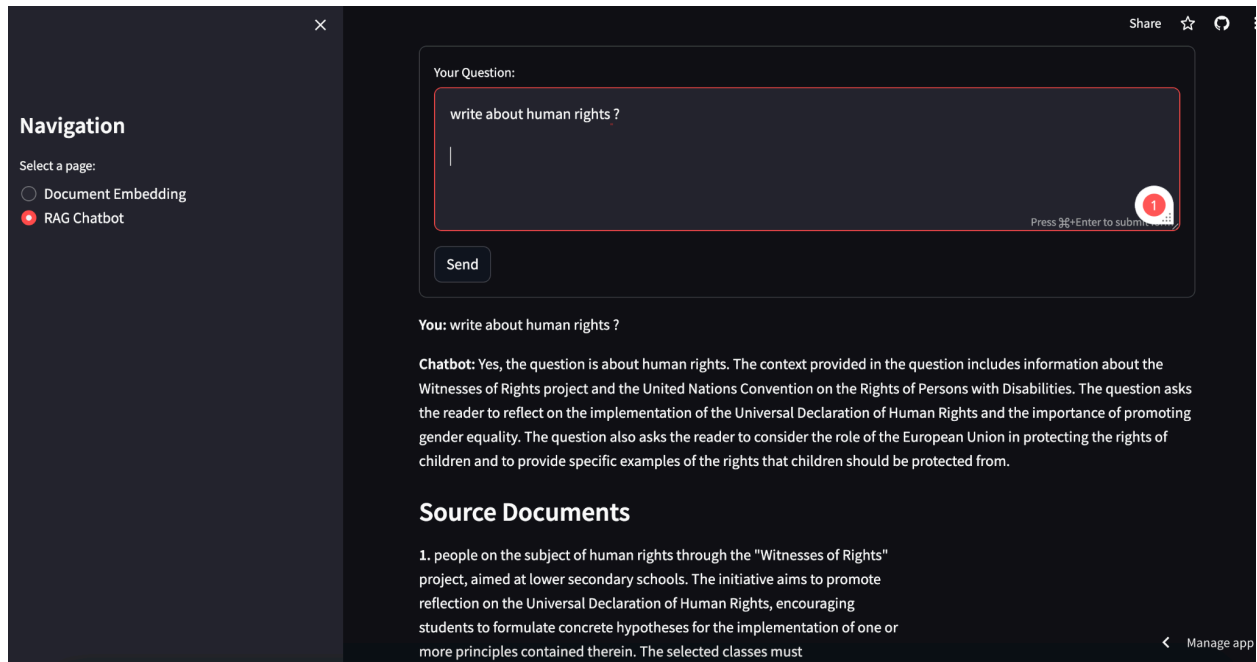
1. people on the subject of human rights through the "Witnesses of Rights" project, aimed at lower secondary schools. The initiative aims to promote reflection on the Universal Declaration of Human Rights, encouraging students to formulate concrete hypotheses for the implementation of one or more principles contained therein. The selected classes must delve deeper into one of the themes touched upon by the Universal Declaration and verify its implementation in their territory, also formulating any proposals to increase compliance with it. How have MPs engaged and inspired young people to address the issue of gender equality? 25 READING SHEETS Among the relevant projects, developed in recent years on the subject of gender equality, the following are worth mentioning: 2010; "being born and dying a woman in contemporary society" by students of a school in San

2. Project Structure

The project consists of the following files and directories:

- `app.py`: The main Streamlit application that handles user interactions, document uploads, and chatbot conversations.
- `chatbot.py`: Contains functions for preparing the RAG pipeline (loading the LLM, embeddings, and vector store) and generating answers.

- `document_processing.py`: Handles reading documents (PDF and TXT), splitting them into chunks, and creating/updating the vector store.
- `vector store/`: This directory stores the vector representations of your documents, making them easily searchable by the chatbot.



3. Setup and Installation

1. Clone the Repository:

2. Bash

git clone <repository_url>

3.

4.

5. Create a Virtual Environment (Recommended):

python -m venv env source env/bin/activate # On Windows, use `env\Scripts\activate`

3. **Install Dependencies:**

```
```bash
```

```
pip install -r requirements.txt
```

#### 4. Obtain a Hugging Face API Key:

- Visit <https://huggingface.co/settings/tokens> and create an access token.
5. **Set Environment Variables:**
- Rename the `.env.example` file to `.env`.
  - Replace the placeholder value `API_KEY` in the `.env` file with your Hugging Face API key.

#### 4. Usage Guide

##### 4.1 Document Embedding

1. **Upload Documents:** Go to the "Document Embedding" page in the Streamlit app.
2. **Select or Create Vector Store:** Choose an existing vector store or create a new one.
3. **Upload your documents:** Select PDF or TXT files to upload.
4. **Click "Save Vector Store":** This will process your documents, create embeddings, and store them in the selected vector store.

##### 4.2 RAG Chatbot

1. **Initialize the LLM Model:**
  - Go to the "RAG Chatbot" page.
  - Expand the "Initialize the LLM Model" section.
  - Enter your Hugging Face API key (this is your token).
  - Select the vector store you created.
  - (Optional) Adjust the temperature and maximum character length.
  - Click "Initialize Chatbot".
2. **Chat with the Bot:**
  - Enter your question in the text area.
  - Click "Send" to get an answer.

- The chatbot will display its response along with the source documents it used to generate the answer.

## 5. Advanced Customization (Optional)

You can customize the chatbot by:

- **Changing the LLM:** Modify the `llm_model` variable in `app.py` to use a different Hugging Face model.
- **Changing the Embedding Model:** Modify the `instruct_embeddings` variable to use a different embedding model.
- **Experimenting with Prompt Engineering:** Try different prompts to see how they affect the chatbot's responses.

## 6. Limitations and Considerations

- The chatbot's performance depends on the quality and relevance of your uploaded documents.
- Large documents may take longer to process.
- The chosen LLM may have limitations in understanding complex questions or generating accurate answers.

## 7. Troubleshooting

- **API Key Not Found:** Ensure you have entered a valid Hugging Face API key.
- **Vector Store Not Found:** Check that the selected vector store exists in the "vector store" directory.

## More in Details of Codes :

### **app.py**

This is the main Streamlit application file, responsible for the user interface and overall control flow of the application.

- Setup and Configuration:
  - Loads environment variables from the .env file using `load_dotenv()`.
  - Sets up the Streamlit page configuration and sidebar navigation.
- Main Function:
  - Uses `st.sidebar.radio` to navigate between "Document Embedding" and "RAG Chatbot" pages.
  - Calls `display_document_embedding_page()` or `display_chatbot_page()` based on user selection.
- `display_document_embedding_page()`:
  - Allows users to upload PDF or TXT documents.
  - Inputs for instruct embeddings model, chunk size, and chunk overlap.
  - Options to create a new vector store or select an existing one.
  - Processes the uploaded documents and stores the embeddings in the specified vector store.
- `display_chatbot_page()`:
  - Provides an interface to initialize the chatbot with settings like API token, LLM model, embeddings model, temperature, and maximum character length.
  - Users can input questions and receive answers from the chatbot.
  - Displays source documents used for generating the response.

### **chatbot.py**

This file contains functions related to preparing the RAG (Retrieval-Augmented Generation) chatbot model and generating answers.

- `prepare_rag_llm(api_key, vector_store_name, temperature, max_length)`:
  - Loads embeddings and the vector store.

- Initializes the language model using the Hugging Face API.
- Sets up a conversation memory buffer.
- Creates a ConversationalRetrievalChain for the chatbot to use.
- `generate_answer(question, token):`
  - Uses the initialized conversation chain to generate a response to the user's question.
  - Returns the chatbot's answer and the source documents used.

### **document\_processing.py**

This file handles the processing of documents, including reading, splitting, and storing embeddings.

- `read_pdf(file):`
  - Reads and extracts text from a PDF file using PdfReader.
- `read_txt(file):`
  - Reads and extracts text from a TXT file.
- `split_doc(document, chunk_size, chunk_overlap):`
  - Splits the document into smaller chunks based on the specified chunk size and overlap using RecursiveCharacterTextSplitter.
- `embedding_storing(split, create_new_vs, existing_vector_store, new_vs_name):`
  - Creates or loads a FAISS vector store.
  - Stores document embeddings in the vector store.
  - Merges with an existing vector store if specified.

### **utils.py**

This utility file contains helper functions, specifically for loading environment variables.

- `load_env_variables():`
  - Loads environment variables from the .env file using `load_dotenv()`.
  - Returns the value of the `API_KEY` environment variable.



