**Proyecto 3 – LLMs**

Andre Gonzalez -739922

Sergio Dueñas - 739300

**1. Descripción General:**

Nuestro objetivo se trata de crear un chatbot que pueda recibir un pdf o documento y lo pueda leer para poder interactuar con el usuario en base al contenido del documento, pudiendo responder preguntas como el contenido del pdf, su autor, resumen, etc.

El archivo **app.py** contiene un programa para construir un chatbot que permite a los usuarios interactuar con el contenido de múltiples archivos PDF. Utiliza la biblioteca Streamlit para la interfaz de usuario, así como varias otras bibliotecas para el procesamiento de texto, la generación de embeddings y la construcción de modelos de conversación.

**2. Flujo Principal:**

* Se configura la interfaz de usuario utilizando Streamlit.
* Se permite al usuario cargar archivos PDF y realizar preguntas sobre su contenido.
* Cuando el usuario hace una pregunta, se procesa utilizando las funciones de procesamiento de texto y se pasa al chatbot.
* El chatbot genera una respuesta utilizando la cadena de conversación y se muestra al usuario.

**3. Dependencias Externas:**

* Streamlit, se utiliza para crear la interfaz de usuario interactiva. En el código, se importa como **streamlit** y se utiliza para crear elementos de la interfaz de usuario, como encabezados, entradas de texto, botones y barras laterales. También se utiliza para mostrar mensajes de estado, como mensajes de carga mientras se procesan los datos.
* Dotenv, se utiliza para cargar variables de entorno desde un archivo **.env**. En el código, se importa como **load\_dotenv** y se utiliza para cargar las variables de entorno, aunque no está explícitamente utilizado en el código proporcionado, es una buena práctica cargar las variables de entorno desde un archivo **.env** para mantener las credenciales y configuraciones fuera del código fuente.
* PyPDF2, se utiliza para trabajar con archivos PDF. En el código, se importa como PdfReader y se utiliza para extraer el texto de los archivos PDF que el usuario carga en la aplicación. La función get\_pdf\_text() utiliza PyPDF2 para leer el texto de los archivos PDF y combinarlos en una sola cadena de texto.
* Langchain, se utiliza para varias tareas relacionadas con el procesamiento del lenguaje natural (NLP), como dividir el texto en fragmentos más pequeños, generar embeddings de texto, construir modelos de conversación y crear cadenas de recuperación conversacional. Sin más información sobre esta biblioteca, no podemos proporcionar detalles específicos sobre cómo se utiliza en el código.
* htmlTemplates, se utiliza para definir el estilo de los mensajes de usuario y bot en la interfaz de usuario de la aplicación. En el código, se importa como css, bot\_template, y user\_template, y se utiliza para mostrar mensajes formateados en la interfaz de usuario utilizando Streamlit.

**4. Funciones Principales:**

**get\_pdf\_text(pdf\_docs)**

* **Descripción:** Esta función recibe una lista de archivos PDF y extrae el texto de cada uno de ellos.
* **Parámetros:**
  + **pdf\_docs**: Lista de archivos PDF.
* **Retorno:** Una cadena de texto que contiene el contenido combinado de todos los archivos PDF.

Texto

Descripción generada automáticamente

**get\_text\_chunks(text)**

* **Descripción:** Divide una cadena de texto en trozos más pequeños para su procesamiento.
* **Parámetros:**
  + **text**: Cadena de texto a dividir.
* **Retorno:** Una lista de trozos de texto.

Texto

Descripción generada automáticamente

**get\_vectorstore(text\_chunks)**

* **Descripción:** Genera un almacén de vectores a partir de los trozos de texto utilizando embeddings de modelos de lenguaje.
* **Parámetros:**
  + **text\_chunks**: Lista de trozos de texto.
* **Retorno:** Un almacén de vectores construido a partir de los trozos de texto.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**get\_conversation\_chain(vectorstore)**

* **Descripción:** Crea una cadena de conversación utilizando un modelo de lenguaje y un almacén de vectores.
* **Parámetros:**
  + **vectorstore**: Almacén de vectores.
* **Retorno:** Una cadena de conversación configurada con el modelo de lenguaje y el almacén de vectores dados.

Texto

Descripción generada automáticamente

**handle\_userinput(user\_question)**

* **Descripción:** Maneja la entrada del usuario y devuelve la respuesta del chatbot.
* **Parámetros:**
  + **user\_question**: Pregunta del usuario.
* **Retorno:** Ninguno (La respuesta se muestra directamente en la interfaz de usuario).

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

**main()**

* **Descripción:** Función principal que configura la interfaz de usuario y controla el flujo principal del programa.
* **Parámetros:** Ninguno.
* **Retorno:** Ninguno.

Texto

Descripción generada automáticamente

**5. Como usarlo:**

* Crear un ambiente virtual
* Dentro del ambiente virtual correr “pip install -r requirements.txt” para poder instalar las versiones usadas
* Crear un .env donde agregues su api key, nombrándola como OPENAI\_API\_KEY= “##”
* Correr en la terminal “streamlit run app.py”

**6. Conclusiones**

Se logró el objetivo, pudimos crear un chatbot que interactué con el usuario a partir del pdf proporcionado. Se pudo desplegar con streamlit pero nos hubiese gustado, para mejorar en un futuro, el despliegue en hugging fase, el cual no se pudo lograr por problemas con las librerías. Del mismo modo, nos hubiera gustado poder implementar un prompt para poder decirle al chatbot como interactuar, ya se de forma amable, seria o incluso actuando como algún personaje.