## Algoritmo para generar preguntas y respuestas a partir de un contexto dado.

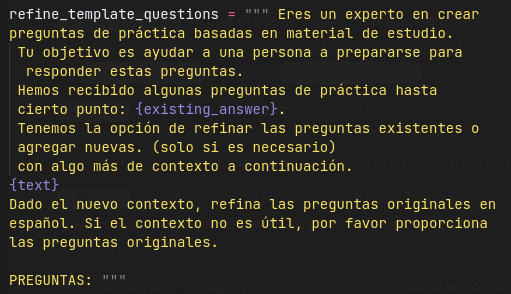
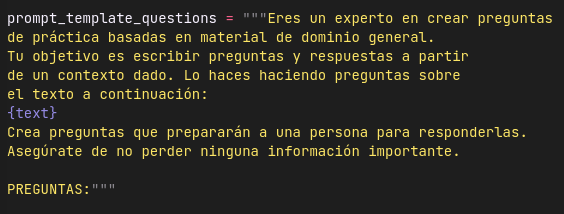
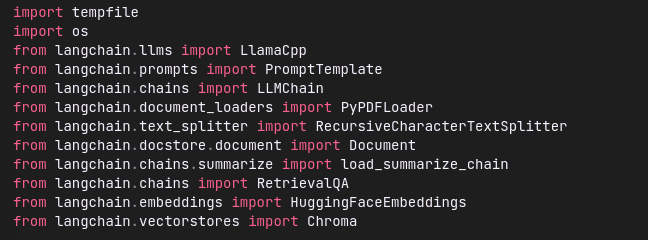
Primeramente se define la localización donde se encuentra el modelo del lenguaje que se va a utilizar.



Posterior a esto, se realizan las importaciones pertinentes. Luego, se define el prompt que se le va a pasar al modelo del lenguaje. Como se puede observar, en el prompt va incluido el contexto que se va a analizar, el cual se declara como la variable "text". La variable "refine\_template\_questions" contiene otro prompt similar al anterior, el cual se va a utilizar para evaluar el resultado que dio el modelo al primer prompt, refinándolo para mejorar así el resultado final.

Después, se declara la variable "PROMPT\_QUESTIONS". Esta variable, mediante la función "PromptTemplate", va a formatear el prompt y definir que la variable de entrada que se le va a pasar, que es el contexto, hace alusión a "text".

Por último, "REFINE\_PROMPT\_QUESTIONS" hace algo similar, pero en este caso, recibe como variables de entrada la respuesta que se dio al prompt anterior y nuevamente el contexto para poder realizar el refinado de las preguntas.



Para poder procesar el PDF que el usuario envíe, se declara la variable "loader", la cual carga el PDF dada la dirección del mismo. Para obtener sus datos, se ejecuta el método "load" de PyPDFLoader.



Luego de cargar los datos, lo primero que se va a hacer es procesarlos para obtener el contenido de cada página, lo que se traduce en realizar una concatenación de strings. El próximo paso es inicializar una variable que permitirá cortar en partes (chunks) el contenido de "text\_question\_gen". En la próxima línea, cuando se tienen los chunks, lo que se hace es definir, a partir de cada chunk, un documento. El documento tendrá el contenido de cada chunk, de esta manera es más organizado a la hora de darle el contenido al modelo para que realice el procesamiento.

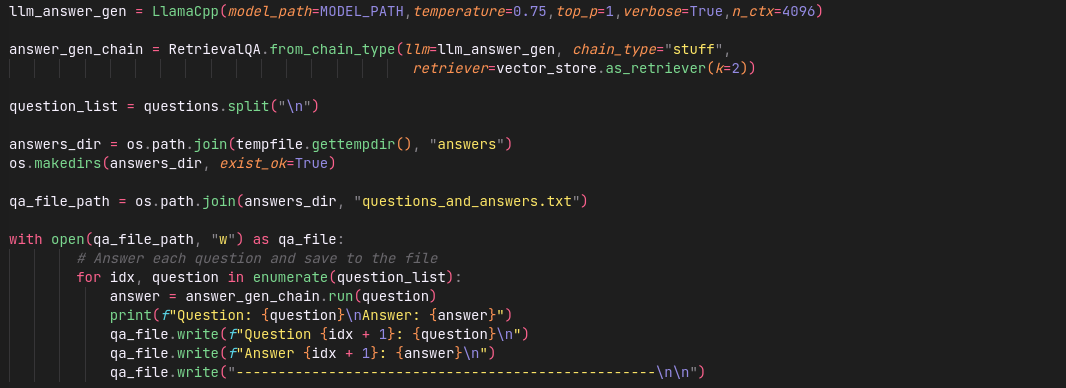
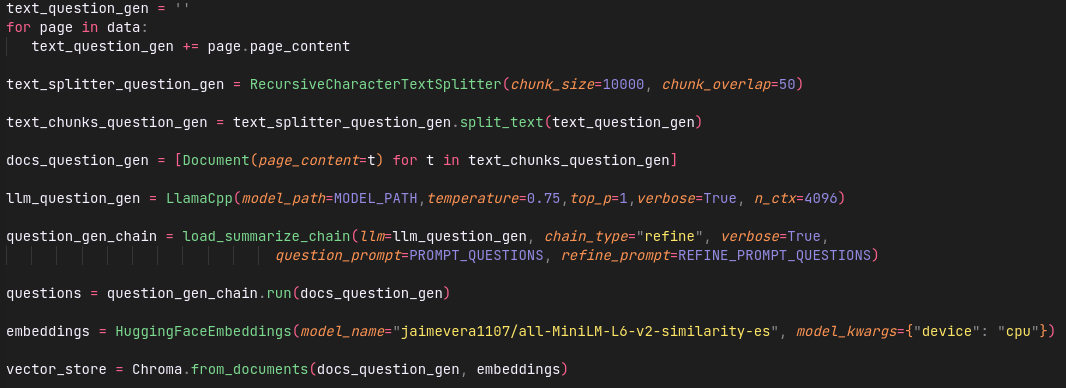
Se define el modelo en la variable "llm\_question\_gen". Esta variable, mediante la función "LlamaCpp", toma la localización del modelo que se había definido previamente y se le pasan los parámetros del modelo. Estos parámetros, luego de varias pruebas, resultaron ser los que mejores resultados arrojaron a la hora de generar preguntas.

A continuación, se define "question\_gen\_chain", la cual se define con la función "load\_summarize\_chain". Lo que hace es cargar una cadena pre-configurada de LangChain para hacer resúmenes y generar preguntas. A esta se le especifica el modelo que va a utilizar, el cual es el que se había declarado previamente. Luego se indica que la cadena que va a utilizar es "refine", para refinar el resultado como anteriormente se había descrito. Se le especifica que sea verboso, lo cual permitirá el envío de mensajes detallados por consola en la ejecución de la cadena. Se le pasa como "question\_prompt" o prompt base "PROMPT\_QUESTIONS" y como "refine\_prompt" "REFINE\_PROMPT\_QUESTIONS".

Después se crea la variable "questions", la cual va a almacenar un arreglo con las preguntas generadas a partir del documento que se había procesado previamente. En la línea siguiente, entonces, se declara el objeto "embeddings" que mapeará texto a vectores numéricos para así poder hallar la similitud de las palabras para su corrección. Se utiliza el modelo "all-MiniLM-L6-v2-similarity-es" de HuggingFace, ejecutándolo en la CPU. Para almacenar estos vectores, se crea una base de datos "vector\_store" usando Chroma, una base de datos para vectores, pasándole el documento y los vectores generados a partir del texto.

Luego se declara el modelo que va a generar las respuestas, el cual tiene la misma configuración que el modelo anterior. Se crea una cadena de QA que utiliza un modelo de generación de respuestas para generar respuestas, una cadena "Stuff" para formatear las preguntas y respuestas, y un retriever de "vector\_store" para recuperar los documentos relevantes de una base de datos. El retriever recuperará los 2 documentos principales para cada pregunta antes de pasarlos al LLM para generar una respuesta.

Después se crea una lista de preguntas, luego de cortarlas donde se encuentre el carácter salto de línea. Próximo a esto, se crea un directorio temporal para almacenar las respuestas generadas. Luego se crea el archivo que va a almacenar las preguntas y respuestas generadas, llamado "questions\_and\_answers.txt". Se abre el archivo txt para después comenzar a iterar sobre la lista de preguntas y así generar las respuestas a partir de la pregunta y el contexto. Por último, se guardan en el archivo las preguntas y respuestas generadas.



## Problemas presentados

Para elegir un modelo que generara las preguntas y respuestas de forma correcta, se realizó una investigación y una prueba con diversos modelos. Estos modelos presentaban consistentemente el problema de que alguna de las oraciones siempre se generaba en inglés, ya que los datos de entrenamiento de la mayoría de estos modelos son en inglés. Este inconveniente no ocurre con el modelo que se emplea, ya que este ha pasado por un proceso de fine-tuning con un dataset en español, lo que le brinda una mejor comprensión del lenguaje.

Otro problema encontrado fue que el modelo más preciso identificado no era compatible con el framework Langchain, dado que su método de cuantización no era uno de los admitidos por dicho framework. Además, este modelo solo podía ejecutarse utilizando una unidad de procesamiento gráfico (GPU).

## Licencia de el modelo utilizado

El modelo base empleado es el Mistral 7B, que se encuentra sujeto a la licencia Apache 2.0. Dicha licencia, reconocida por su flexibilidad, fomenta la libertad en el uso, modificación y distribución del software de código abierto. En virtud de lo expuesto anteriormente, no existen restricciones que impidan la utilización de este modelo en el presente proyecto.