Laura Sofía Vera – 201123145
Juan Camilo Arrieta – 202121839
Juan Pablo Camacho -- 202110977
Daniela Herrera – 202113704

RAG: Recetas colombianas

Para nuestro trabajo, escogimos un RAG de recetas colombianas. Este se cargaría con información de múltiples recetas, reseñas de recetas, libros de cocina, fichas de ingredientes típicos y técnicas culinarias; así como, variaciones de las recetas por región para que tenga distintas versiones y pueda satisfacer las necesidades de los usuarios. Con esta base de datos tendrá suficiente corpus para poder generar las recetas reales a pedido de los usuarios. Por ejemplo, si alguien quiere un ajiaco sin papa pastusa, podrá obtener varias opciones de receta según la región y con la modificación solicitada.

Ahora bien, el modelo tiene algunas limitaciones. En principio, este puede llegar a alucinar con *prompts* como "dame la mejor receta de ajiaco" o "recomiéndame algo que me guste", ya que no tiene la información necesaria para determinar los gustos del usuario; y, aunque tenga las reseñas en sus datos de búsqueda, la "mejor receta" sigue siendo un *prompt* muy subjetivo para este tipo de modelo. Adicionalmente, depende demasiado del tamaño del corpus, si este es muy pobre las respuestas serán deficientes y tenderá a alucinar más si no encuentra la información pertinente. Por otro lado, como solo es un modelo de *retrieval*, las recomendaciones tendrán limitaciones como: no aprender los gustos del usuario, no saber que elementos hay disponibles para la receta, no valida sabores reales, entre otros. Así, requiere de mucha información y *prompts* precisos para poder dar respuestas optimas.

Finalmente, para la realización del RAG utilizamos una base de datos de Hugging Face con recetas de varias partes de mundo. Dado que, por tiempo, fue difícil conseguir una base de datos solo con recetas colombianas.

A continuación, detallamos los pasos realizados en el desarrollo del RAG:

1. Encoder

Primero cargamos la colección completa de recetas desde el repositorio escogido de Hugging Face. Luego reducimos el volumen de los datos, pues hicimos varios intentos con la base completa y utilizando varios modelos diferentes, pero el código no nos terminaba nunca de cargar. Por esto se seleccionaron solo las primeras 2000 recetas para acelerar las pruebas y la creación de *embeddings*. Sin embargo, se debe tener en cuenta que con esto se disminuye la

precisión de las respuestas y se reduce la variedad de recetas en el RAG. Finalmente, limpiamos el texto eliminando saltos de línea y espacios múltiples.

Luego para la creación de los *chunks* se inicializa el *splitter* escogiendo como *chunk size* 512 y *chunk overlap* de 150. Esto debido a el modelo de *embedding* está entrenado con una secuencia máxima de 512 tokens. Luego cada receta es segmentada en fragmentos pequeños, cada fragmento se añade a una lista y se le asigna el índice de la receta original y índice del fragmento dentro de la receta.

A continuación, vectorizamos la base de datos y aplicamos el modelo encoder: **MiniLM**. Este es un modelo SBERT multilingüe, ligero, rápido y gratuito de Hugging Face que mapea frases y párrafos con asignación de un espacio vectorial denso de 364 dimensiones. Dado un texto de entrada, el modelo genera un vector que captura la información semántica y puede utilizarse para recuperar información o agruparla.

Utilizamos este modelo porque, al ser un SBERT, es el tipo de modelo de Python ideal para entrenar modelos de recuperación y similitud de textos. Además, sus intenciones de uso, descritas por los desarrolladores, son codificar oraciones y párrafos cortos, lo cual es perfecto para textos cortos y precisos como lo son las recetas (*Sentence-Transformers/All-MiniLM-L6-v2 · Hugging Face*, 2024).

Entonces, implementamos un código que acelerará la vectorización con un *batching* de 128. Así, en lugar de procesar un *chunk* a la vez, procesa 128 *chunks* simultáneamente, maximizando la eficiencia y reduciendo el tiempo de ejecución. Luego, creamos la base de datos vectorial FAISS, que toma todos los *chunks* y sus metadatos para convertirlos en vectores con el encoder y los almacena en una estructura optimizada para búsqueda. Con esta base de datos generamos un índice vectorial de FAISS que, de acuerdo con la función principal de consulta, recupera los documentos más importantes según la consulta del usuario. De esta manera, el sistema vectoriza la pregunta y busca los *k* vectores de *chunks* más cercanos en FAISS. Retornando el *chunk* de texto y un puntaje de similitud, que indica qué tan bien coincide con la pregunta.

Finalmente, el modelo muestra los resultados, nos da: la pregunta, el puntaje de relevancia, un fragmento del contenido de las recetas relevantes y los metadatos que indiquen de dónde proviene la información (el id del documento o el *chunk*).

2. Decoder

Para la etapa de *decoder*, implementamos un sistema híbrido que combina la recuperación local de información mediante FAISS con un modelo de lenguaje avanzado alojado en la nube. Este enfoque permite que el RAG no solo encuentre los fragmentos más relevantes en la base vectorial, sino que también genere una respuesta coherente, estructurada y contextualizada utilizando un modelo instructivo de mayor capacidad.

En primer lugar, construimos una función de recuperación llamada `retrieve_recipes()`, la cual opera directamente sobre la base vectorial generada en la etapa del encoder. Esta función recibe la consulta del usuario, vectoriza el *query* utilizando el mismo modelo de embeddings y recupera los *k* chunks más similares empleando la función `similarity_search_with_score`. De esta forma, el modelo puede identificar los fragmentos más pertinentes y acompañarlos con un puntaje de similitud, lo que facilita interpretar qué tan relevante es cada chunk para la consulta. Adicionalmente, la función imprime el contenido inicial del fragmento y sus metadatos, como el identificador de la receta original y el número del chunk, con el fin de mantener trazabilidad sobre el origen de la información.

Posteriormente, desarrollamos un pipeline completo denominado `rag_pipeline_cloud()`, que integra el *retrieval* local con FAISS, y la decodificación mediante un modelo de lenguaje alojado en la nube. En el primer paso, la función ejecuta la recuperación mediante FAISS y compila los tres chunks más relevantes en un único bloque de contexto. Cada fragmento se formatea y une en un texto continuo que posteriormente se le entrega al modelo generativo. Esta estructura garantiza que el modelo en la nube reciba únicamente información precisa y localizada, manteniendo el funcionamiento clásico de un RAG: el LLM no inventa la información (no alucina), sino que la desarrolla en función de lo recuperado por el buscador vectorial.

Para la etapa de decodificación se utiliza el modelo Mistral-7B-Instruct, accedido mediante la API de Hugging Face. Este modelo se escogió por su capacidad para seguir instrucciones de manera precisa, su eficiencia computacional y su buena performance en tareas de transformación, explicación y reformulación de textos. Dada su ejecución remota, se evita saturar la memoria del entorno local, permitiendo obtener respuestas de alta calidad.

A continuación, se construyó un *prompt de sistema* que define el rol del modelo como un chef experto bilingüe. Dentro de este prompt se establecen instrucciones claras para guiar la generación, incluyendo reglas estrictas de traducción y la obligación de presentar las respuestas en español, con título, ingredientes y pasos de preparación. Este enfoque de *prompt engineering* garantiza uniformidad en las respuestas y reduce el riesgo de errores semánticos de traducción o interpretación.

Finalmente, el pipeline envía el contexto recuperado y la pregunta del usuario al modelo Mistral en la nube, generando así una respuesta completa que explica la receta solicitada, traducida y reescrita en un formato accesible y estructurado. De esta forma, la etapa de decodificación permite cerrar el ciclo del RAG, combinando la precisión del retrieval local con la expresividad del modelo generativo, asegurando respuestas más útiles, coherentes y ajustadas a las necesidades del usuario.

3. Conclusiones

El sistema RAG desarrollado integra de manera efectiva un proceso de recuperación local con un modelo generativo en la nube. En la etapa del encoder, se construyó una base vectorial a partir de una colección de recetas, aplicando un modelo de embeddings adecuado para textos cortos y segmentando la información en chunks que facilitan una recuperación más precisa. El uso de FAISS permitió optimizar las búsquedas y obtener rápidamente los fragmentos más relevantes para cada consulta.

En la etapa del decoder, se implementó un pipeline que combina los resultados obtenidos por el vector store con un modelo instruccional avanzado (Mistral-7B-Instruct). Este modelo recibe los chunks recuperados y produce una respuesta final organizada, en español y con el

formato solicitado por el usuario. Gracias al uso de un LLM externo, el sistema logra generar texto claro y coherente sin depender de la capacidad computacional del entorno local.

En conjunto, el proyecto muestra cómo un RAG puede construirse de manera modular: primero recupera información relevante mediante embeddings y FAISS, y luego la transforma en una respuesta completa utilizando un modelo generativo. Aunque el corpus empleado no es exclusivamente de recetas colombianas, el pipeline funciona correctamente y permite entregar respuestas útiles a partir del contenido disponible.