Modelos de Lenguaje - Informe Final Grupo 4

Asistente Conversacional RAG: Recetario Saludable

Tema elegido para el asistente conversacional:

El asistente conversacional desarrollado responde consultas sobre recetas saludables para toda la familia. Está basado en el recetario “100 recetas saludables para disfrutar en familia”, con el objetivo de acercar ideas de cocina accesibles, variadas y nutritivas a través de una experiencia conversacional.

Justificación de la elección del tema:

El recetario fue elaborado por profesionales de la salud con enfoque en nutrición infantil y alimentación saludable de la Asociación Española de Pediatría de atención primaria. La temática es socialmente relevante por su foco en el bienestar familiar, y técnicamente desafiante por requerir un parseo no estructurado de PDF y una recuperación semántica precisa. Además, el dominio de recetas permite evaluar la capacidad del sistema para estructurar respuestas claras y prácticas.

Fuentes utilizadas para construir la base de conocimiento

- Archivo PDF: *recetas_fys.pdf* (proporcionado en formato digital)
- Contenido completamente extraído del PDF, sin fuentes externas adicionales

Preprocesamiento y parseo de datos

Se utilizó *pdfminer.six* para extraer el texto entre las páginas 13 y 121 del recetario que eran las que contenían las recetas. Mediante expresiones regulares se detectaron los campos:

- Título
- Ingredientes
- Elaboración
- Condiciones especiales (ej. “apta para celíacos”)
- Autora

El texto fue limpiado eliminando encabezados, pies de página y líneas vacías. Cada receta se estructuró como un documento con page content y metadatos relevantes.

Modelo LLM utilizado

Inicialmente se probaron modelos locales usando Transformers:

- **Gemma 2B** → Baja calidad de respuesta, no obteniendo una buena recuperación.
- **LLaMA-3.2-3B** → Respuestas mejores pero requerimientos de memoria muy altos.
- **Mistral-7B** → Respuestas mejores pero requerimientos de memoria muy altos.

Finalmente, se optó por **gpt-3.5-turbo** usando Chat OpenAI a través del framework LangChain, debido a su estabilidad, calidad de respuesta y facilidad de integración.

Sistema de embeddings

Se utilizó el modelo intfloat/multilingual-e5-large, vía HuggingFaceEmbeddings, por su buen rendimiento en tareas semánticas multilingües. Las recetas fueron embebidas y almacenadas en un índice FAISS, configurado para recuperación por similitud coseno (k=4), por lo que busca las 4 recetas más parecidas dentro del recetario. Estas se seleccionan no por coincidencia exacta de palabras, sino por similitud en el significado. Luego, estas 4 recetas se usan como contexto para que el modelo genere una respuesta relevante y fundamentada.

Vectorización

Optamos por FAISS porque es eficiente para realizar búsquedas vectoriales por similitud. Tiene un buen rendimiento con gran cantidad de vectores, y funciona en CPU. Además, se integra fácilmente con LangChain, lo que simplifica la construcción del pipeline RAG sin requerir servidores externos como en el caso de Elasticsearch. Esto lo convierte en una buena opción para sistemas que deben correr localmente o en entornos controlados.

Diseño del sistema RAG

- **Arquitectura general**: FastAPI + LangChain + FAISS + OpenAI + Streamlit
- **Flujo**:

    1. Usuario envía pregunta vía interfaz o API
    2. Sistema recupera recetas similares con FAISS
    3. Se construye un prompt con contexto + pregunta
    4. El LLM genera una respuesta

- **Prompting**: El prompt fue diseñado para guiar al modelo a responder únicamente en base a la información extraída del recetario, evitando que invente o se desvíe del contexto. Se indica al modelo que mantenga un tono cálido y claro, que redacte siempre en español, que presente las recetas de forma separada y estructurada, y que cierre cada respuesta alentando a cocinar en familia y seguir consultando el recetario.
  Versión optimizada de prompt que utilizamos:

“Eres un asistente especializado en cocina saludable para familias.Se te proveen fragmentos de varias recetas. Tu tarea es identificar las recetas relevantes a la pregunta y listar sus títulos, ingredientes y pasos por separado.Además, de proveer las recetas, debes dar un breve comentario al inicio sobre la pregunta del user o las recetas provistas.Debes dar una respuesta clara, amable, útil y en español. No mezcles información de recetas diferentes.La respuesta debe estar basada únicamente en la información del contexto provisto.Si no sabes la respuesta porque no se encuentra en los fragmentos del contexto dado, responde con \"No lo sé, no forma parte del recetario\" y absolutamente nada más. No inventes ingredientes, pasos, ni consejos que no estén presentes en el recetario. Siempre finaliza alentando a cocinar en familia, comer saludable y consultar el recetario completo.”

Capturas de pantalla del sistema (Streamlit)

Resumen del backend

- rag_pipeline.py

Este módulo se encarga de todo el procesamiento de datos:

- Carga el PDF con recetas.
- Extrae y limpia el texto de cada receta.
- Vectoriza los documentos usando embeddings de Hugging Face.
- Construye un índice FAISS para recuperación por similitud.
- Configura el pipeline completo de RAG que combina recuperación + generación con un modelo LLM de OpenAI.

Este pipeline queda expuesto como una clase (RAGPipeline) lista para ser usada desde otros módulos.

- main.py

Define una API utilizando **FastAPI**. Expone un endpoint que recibe una pregunta en formato JSON, la procesa con el RAGPipeline, y devuelve una respuesta generada por el modelo.

Así, permite que otros componentes (como la interfaz web o clientes externos) se comuniquen con el sistema de forma simple y estructurada.

- run_api.sh y setup_api.sh

Son scripts bash que:

- Preparan el entorno local (setup_api.sh, por ejemplo creando un entorno virtual).
- Ejecutan la API y la interfaz web (run_api.sh lanza FastAPI y Streamlit en paralelo).

- OpenAI.env

Contiene la clave de API de OpenAI usada por el modelo GPT (gpt-3.5-turbo) para la generación de texto.

- run_docker.sh

Permite ejecutar toda la aplicación de forma contenedorizada mediante Docker, lo que facilita su despliegue y portabilidad.

- Conexión entre componentes

1. rag_pipeline.py contiene la lógica del sistema RAG y se importa dentro de main.py.
2. main.py expone esa lógica como una API REST con FastAPI.
3. run_api.sh lanza tanto FastAPI como Streamlit para probar la app con interfaz.
4. El archivo .env es cargado automáticamente para habilitar el uso del modelo de lenguaje.
5. Toda la aplicación se puede correr localmente (setup_api.sh y run_api.sh) o vía Docker (run_docker.sh).

Conclusiones

Este proyecto permitió incorporar aprendizajes sobre cómo construir un sistema RAG de punta a punta. A partir de una fuente como un recetario en PDF, se logró desarrollar un asistente capaz de responder preguntas de forma coherente y contextualizada, sentando una base para poder aplicar esta misma lógica en otros temas en el futuro.

Uno de los principales logros fue entender cómo funciona el flujo completo de recuperación y generación de información, lo que resultó muy útil para comprender mejor cómo los modelos de lenguaje pueden utilizar documentos externos como apoyo. También pudimos experimentar con distintos modelos, como Gemma, Mistral o LLaMA, y observar en la práctica sus limitaciones y necesidades técnicas, como también poder evaluar su performance, lo que nos llevó a tomar decisiones sobre qué modelo usar y por qué.

En términos de infraestructura, aprender a usar Docker fue clave. Esta herramienta nos permitió encapsular todo el sistema en un entorno que se puede ejecutar fácilmente en cualquier computadora, sin depender de configuraciones locales ni sufrir errores por diferencias entre máquinas.

Por último, otro desafío que tuvimos fue el parseo del documento original. No solo hubo que interpretar un PDF extenso y con estructura irregular, sino que también fue necesario detectar qué partes eran útiles para el asistente y cómo organizarlas. Este proceso nos ayudó a desarrollar un criterio para identificar patrones relevantes dentro de textos complejos.

Como oportunidades de mejora del proyecto, se podría expandir la base de conocimiento con otros recetarios, incorporar filtros para búsquedas más específicas (por ingrediente o condición de salud), y sumar mecanismos de retroalimentación por parte del usuario. A su vez, se podría iterar para que el asistente tenga memoria y pueda generar una conversación.