Get Talent 2025 - Challenge final

Diseño de una API basada en RAG para soporte informativo clínico. Aplicación en oftalmología sobre patologías retinales prevalentes en adultos

Introducción

En el ámbito clínico abunda la información sobre diferentes criterios para diagnosticar patologías o diferentes protocolos para tratarlas. Además, cada médico adecúa los mismos según su experiencia y posibilidades económicas, haciendo que el espectro de criterios cambie de un especialista a otro. Para hospitales o instituciones de salud esto puede ser una dificultad, ya que el mismo paciente atendido por dos médicos diferentes puede recibir un mensaje distinto e incluso erróneo ante la misma consulta, aumentando la incertidumbre sobre su salud y disminuyendo la confianza en los médicos y la institución. Además, estos centros de salud en general forman a futuros médicos especialistas en diferentes áreas, por lo cual es de gran importancia para ellos poder adoptar el mismo criterio con el que trabajan sus superiores y la institución.

Existen guías y protocolos elaborados por expertos de diferentes asociaciones académicas a nivel mundial, quienes recopilan, unifican y actualizan anualmente toda la información existente hasta el momento. Por este motivo, suelen ser manuales extensos y leerlos a cada momento se vuelve una tarea difícil en la práctica diaria. Un dato no menor es que siempre se encuentran escritos en inglés, lo cual puede ser una barrera hacia aquellos que no dominan el idioma.

De esta manera surge la necesidad de unificar criterios en equipos médicos y de ofrecerles acceso rápido y simple a material confiable, estandarizado y minuciosamente seleccionado.

Objetivo

El objetivo de este trabajo es diseñar una API multilenguaje basada en Retrieval Augumented Generation (RAG) de asistencia a médicos oftalmólogos que:

- Su única fuente de información sea información oficial de calidad cuidadosamente seleccionada;
- Provea respuestas estandarizadas y reproducibles;
- Permita preguntas y respuestas en español;

- Ofrezca respuestas confiables alineadas a guías internacionales validadas por consenso entre pares.

Usuarios

La aplicación está pensada principalmente para médicos oftalmólogos que deben tomar decisiones clínicas de diagnósticos y tratamientos sobre diferentes enfermedades, médicos residentes de oftalmología les resultaría útil tanto al momento de tratar con pacientes durante su formación como también al estudiar y finalmente el centro de salud que la implementa aunque no sea usuario directo, ya que aumenta la calidad del servicio que brinda y se posiciona a la vanguardia al aplicar soluciones tecnológicas con Inteligencia Artificial.

Fuentes de información

La *American Academy of Ophtalmology (AAO)* es la institución de mayor prestigio en el área de la oftalmología a nivel mundial. Un comité multidisciplinario elabora guías y protocolos sobre todas las patologías oculares descriptas hasta la actualidad, los cuales se actualizan anualmente o cuando surgieran modificaciones. Estos manuales denominados *Preferred Practice Patterns* (PPP), son los más adoptados por los médicos tanto para diagnosticar como para tratar a los pacientes. En este trabajo se usó como fuente de información verificada uno de los PPP de libre acceso en la página web de la AAO, puntualmente sobre la patología Degeneración Macular Asociada a la Edad.

Desarrollo

Entorno virtual

Se creó y activó un entorno virtual (venv) con las librerías: PyMuPDF, langchain, langchain-community, cohere, python-dotenv, chromadb, fastapi, uvicorn.

Procesamiento previo del texto

Las guías utilizadas como fuentes de información se encuentran en formato .pdf por lo que es necesario pasarlas a .txt para luego generar los chunks.

Se utilizó la librería **PyMuPDF** que respeta mejor que otras librerías los formatos de tablas y columnas dentro de los archivos PDFs.

Se definió la función **extract_text_from_pdf** para extraer el texto del PDF dejando la información del número de página. Esto es necesario para luego citar la información cuando el chat de la respuesta.

También se definió la función **process_all_pdfs**, la cual aplica extract_text_from_pdf a todos los PDFs de la carpeta de origen.

Chunking

Esta etapa se realizó utilizando **RecursiveCharacterTextSplitter** de LangChain. Esta herramienta mantiene el sentido semántico del texto dividiendo desde segmentos grandes hasta caracteres de manera progresiva, de mayor a menor texto.

Embeddings

Se realizaron con Cohere. Se usó **embed-multilingual-v3.0** ya que esto permitirá que las preguntas que haga el usuario encuentren similitudes vectoriales independientemente del idioma de los textos de origen. En general serán preguntas en español, base de datos en inglés y respuestas en español.

Los embeddings se almacenaron en una base de datos ChromaDB. La similitud entre vectores se calculó mediante *coseno*.

A diferencia de lo trabajado en los challenges previos, se agregó la herramienta de persistencia. Esto permite que una vez finalizado el preprocesamiento de los datos de entrada estos se puedan almacenar en la base de datos de manera permanente sin tener que volver a realizar los pasos descritos hasta ahora cada vez que iniciemos la app.

RAG

- RERANKED. Se implementó en la arquitectura del RAG esta herramienta que permite buscar un gran número de vectores similares definido como top_n y luego quedarse con los mejores, definidos como top_k. En archivos largos como el de este proyecto en los que algunos conceptos aparecen en todo el texto y no en un subtítulo único, esta herramienta permite abarcarlos (aunque no a todos) y poder dar una respuesta final con un mejor contexto mejorando así la etapa retrieve del RAG. Definimos **reranked_retriever** con top_n= 20 y top_k=5. También se usó embed-multilingual-v3.0. Lo aplicamos a la pregunta (query).
- RAG. En este archivo se constituye el correcto funcionamiento del RAG: ingresa la pregunta del usuario, aplica reranked_retriever (embedding de la query, busca en la BDD Chroma, rankea con Cohere los fragmentos con mayor similitud en la base de datos y devuelve los 5 más similares), crea contexto con ellos y finalmente se genera la respuesta con el LLM de Cohere según el prompt diseñado para esta aplicación. Se utilizó el modelo **command-r-plus-08-2024** y temperatura= 0,0, ya que al tratarse

de textos médicos se necesita que las respuestas sean iguales y reproducibles ante una misma pregunta.

API

- SCHEMAS. Se define que tanto en la pregunta del usuario como en la respuesta de rag_answer los datos serán tipo string.
- ORQUESTADOR. Generalmente, los usuarios de chatbots usamos palabras conversacionales comunes como saludos o agradecimientos. Para evitar que ante esa entrada el modelo entre al circuito de respuesta del RAG, se implementó esta mejora que genera respuestas predeterminadas para algunas palabras o frases mediante la función **get_generic_responses**. Si el usuario ingresa datos a la app, la función **orchestrated_answer** primero intentará dar las respuestas genéricas, pero si no puede porque la consulta es diferente, entonces responderá con rag_answer.
- ROUTER. En este archivo converge todo lo trabajado, ya que se indican las rutas, se definen los métodos de la api y las funciones que se deben ejecutar cuando llega una pregunta del usuario. La api se realizó con FAST API. El método POST ingresa la query del usuario en el formato definido por schemas e intenta responder por orchestrated_answer. De lo contrario, da error de sistema.
- API. Este es el archivo que luego se ejecutará con uvicorn. Se define la app, se incluyen las rutas definidas en router, y se define el método GET para comprobar que el servidor se levantó.

FRONTEND

Con IA (claude.ia) se diseñó un chat atractivo para usuarios con CSS, JavaScript y HTML.

Puntos de mejora

- Aumentar la base de datos con otros protocolos sobre otras enfermedades oculares diferentes, ampliando la aplicabilidad de la API.
- Refinar el orquestador para que, ante una pregunta fuera de contexto, responda el LLM (implicando un segundo prompt) aunque sin pasar por el RAG. A diferencia de la solución actual predefinida sólo a saludos, agradecimientos o ayuda, esto mejoraría todas las respuestas que estén fuera de contexto.
- Guardar el historial de conversaciones es útil como métrica de control de calidad para el administrador o dueño del centro que implemente la API.

- Mejorar el front end con el uso de filtros cuando haya mas de un protocolo. Así la búsqueda vectorial resulte más rápida y con menos variabilidad.