Trabajo final 2025

Taller Modelos de Lenguajes

UNIVERSIDAD DE MONTEVIDEO

Felipe Renom

Natalia Ripa

Valentina Tedesco

Introducción

El Trabajo Final del Taller de Modelos de Lenguajes tiene como objetivo desarrollar una aplicación del estilo chatbot sobre lenguaje Python que permita realizar consultas sobre el Derecho Uruguayo, buscando brindar una solución funcional, accesible y adaptada al ámbito jurídico nacional, basada en un modelo de lenguaje LLM con arquitectura RAG, usando las herramientas de FastAPI, Streamlit y Docker.

Tema

El tema elegido para el asistente conversacional es el Derecho Uruguayo. La elección del tema se debe a que el Derecho es un tema de suma importancia ya que establece reglas claras sobre cómo deben comportarse las personas en sociedad, resolviendo conflictos y asegurando el orden.

En Uruguay, las leyes están diseñadas para regular relaciones entre individuos, empresas y el Estado, cubriendo una gran cantidad de áreas como el Derecho Penal, Civil, Laboral, de familia, entre otros, lo cual permite que haya una convivencia más armoniosa, evitando el caos o la arbitrariedad.

Un chatbot especializado en leyes de Uruguay tiene los siguientes beneficios:

- Acceso rápido y directo a la información: El Derecho es un campo muy técnico. En muchos casos las personas no tienen conocimientos profundos sobre leyes. Un chatbot especializado en Derecho Uruguayo puede ayudar a que cualquier persona tenga acceso rápido a información relevante sobre normas, derechos y obligaciones. Lo cual facilita la accesibilidad a la justicia y a la información, incluso para personas sin experiencia previa en el ámbito legal.
- Ahorro de costos: El proceso de consulta legal tradicional puede ser costoso y llevar tiempo. No siempre es necesario contratar a un abogado para obtener respuestas básicas o información general.
- Actualización de la legislación: Las leyes pueden cambiar o ser modificadas con frecuencia. Un chatbot puede ser actualizado constantemente para reflejar estos cambios y asegurar que los usuarios estén recibiendo información legal actualizada.
- Accesibilidad: El Derecho puede ser percibido como algo complejo por una gran parte de la población. Un chatbot permite a los ciudadanos acceder a explicaciones claras sobre conceptos legales, resoluciones judiciales o los pasos a seguir en determinados procesos, independientemente de su conocimiento sobre el tema.
- Soporte en múltiples áreas: Uruguay tiene una legislación bastante rica que cubre desde el Derecho Constitucional hasta el Derecho Penal, pasando por el Derecho Laboral, Derecho de Familia, y el Derecho Tributario, entre otros. Un asistente legal puede ofrecer información en diferentes áreas del Derecho, proporcionando una orientación más ajustada a cada necesidad.
- Ayuda a los abogados y estudiantes: Este tipo de tecnología también podría ser útil para abogados y estudiantes que necesiten consultar rápidamente artículos de la ley o buscar información en temas legales específicos.
- Impulso de la digitalización: La digitalización de los servicios legales es una tendencia global, y un chatbot especializado en Derecho es un paso en la modernización del sistema legal y facilita el acceso a la justicia de una manera más eficiente.

Fuentes

Utilizamos como referencia la página web de IMPO, la Dirección Nacional de Impresiones y Publicaciones Oficiales, que es la institución encargada de difundir y dar a conocer la normativa jurídica en Uruguay.

Como fuente de información utilizamos los links de normativas de diferentes áreas del Derecho. La información de los links corresponde a enlaces oficiales de IMPO del Programa Lenguaje Ciudadano. Este programa propone fomentar el conocimiento de la normativa nacional por medio de la traducción a un lenguaje llano y simple de la misma, en el entendido que solo con información calificada sobre esta los ciudadanos pueden ejercer plenamente sus derechos y honrar sus obligaciones.

Los links utilizados como fuente son los siguientes:

- Defensa al Consumidor: https://www.impo.com.uy/derechosconsumidor/
- Licencias Especiales: https://www.impo.com.uy/licenciasespeciales/
- Residuos: https://www.impo.com.uy/gestion-de-residuos/
- Inclusión Financiera: https://www.impo.com.uy/inclusionfinanciera/

Preprocesamiento y parseo

Para la parte de preprocesamiento y parseo, nos encontramos frente al problema de que el HTML de IMPO está todo escrito como , y para los títulos los puso en bold, en vez de headings, entonces optamos por la opción de sacar toda la información dentro del div "wpb_text_column wpb_content_element", y ahí pudimos extraer todo el contenido que necesitamos.

Como está estructurado en un sistema de preguntas y respuestas, hicimos una expresión regular que identifico los signos de interrogación “¿?” para poner eso dentro de la clave “question” en el JSON, y lo que sigue hasta el siguiente signo de interrogación en la clave “answer”.

Hicimos también otra alternativa no tan dinámica y escalable, que fue copiar y pegar en un Word el contenido con headings y párrafos, correctamente estructurado, luego sacamos eso y lo pusimos dentro de un JSON.

En el notebook preprocess.ipynb, dónde se parsean estos Word usando la librería docx para leer los documentos, y la librería os para manipulación de directorios. Con esto, lo que hicimos fue extraer texto del Word y crear un archivo .json para cada link.

Dentro del JSON, lo que hicimos fue poner metadata, como el nombre de la ley, y el link. Luego, pusimos los bloques de preguntas respuestas en el JSON.

Cabe destacar, que al principio de este proceso intentamos poner los 4 links en un solo documento JSON, y el chatbot no respondía bien porque no tenía suficiente contexto.

Modelo LLM

Usamos el modelo de gpt-3.5-turbo, ya que el LLM no tiene una tarea demasiado compleja, entonces para tener menos latencia, preferimos tomar un LLM más ligero que un gpt-4o por ejemplo.

Además, por un tema de facilidad de acceso ya que teníamos acceso a créditos de OpenAI, entonces nos resultó más accesible y fácil usar modelos de OpenAI que recurrir a un Llama 3.2 cómo fue aconsejado en clase.

Sistema de embeddings

Empezamos probando el modelo de embeddings de HuggingFace, 'all-MiniLM-L6-v2', pero cuando para ciertas consultas no lograba vectorizar correctamente algunos textos, y algunos chunks aparecían vacíos. No logramos identificar si el modelo de embeddings era la causa o algún otro factor de nuestro script. Dado esto, decidimos cambiar al modelo de OpenAIEmbeddings.

Adicionalmente, en esta primera prueba comenzamos usando la base de datos vectorial 'ChromaDB'; pero nos saltaba el error mencionado anteriormente. Entonces decidimos cambiar la base vectorial a FAISS.

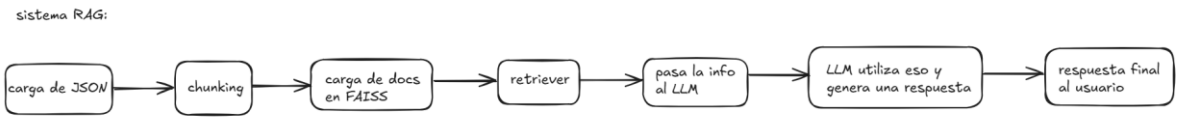
Entonces dado este problema (que algunos chunks contenían la pregunta, pero la respuesta estaba vacía), decidimos cambiar el modelo de embeddings al de OpenAI y la base vectorial a FAISS.

RAG

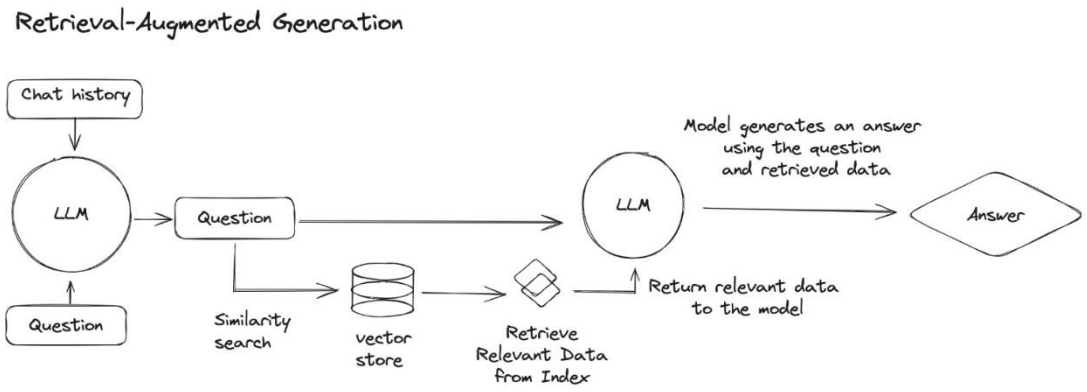
Este sistema RAG sigue los siguientes pasos:

1. Carga los documentos de los JSON y los va asignando a la variable "docs".
2. Hace el Split de los documentos en chunks, acá usamos un chunk_size de 5.000 para asegurarnos que todos los chunks tengan la pregunta y la respuesta completa, que no se parta en dos.
3. Luego se crea la vectorstore de FAISS
4. Se crea el retriever. (hicimos una v2 de esta función para que también acceda al link, que está en la metadata de los JSON).
5. Con la información recuperada, que el LLM (gpt-3.5-turbo) responda en lenguaje natural al usuario.

De manera más gráfica, sería algo así:



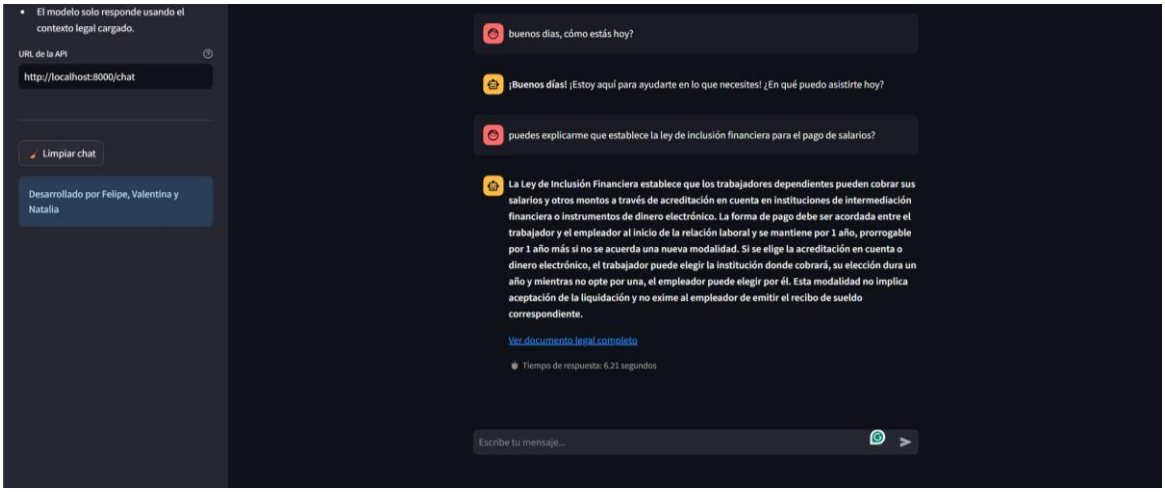
Al mismo tiempo, encontramos esta imagen que resume de forma genérica como hace un RAG, con el cual coincidimos y adaptamos a nuestros requisitos.



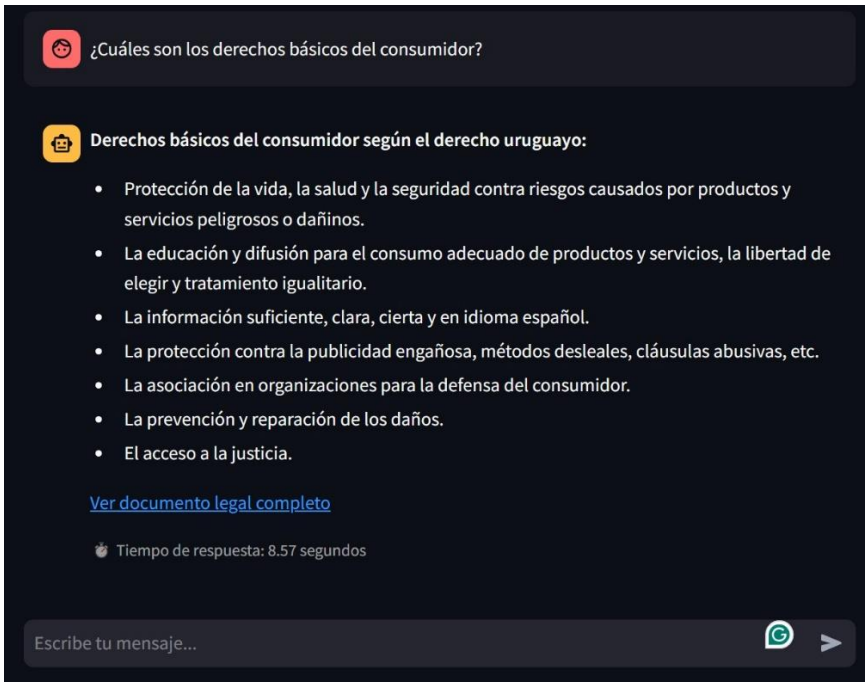
Streamlit

Saludo inicial en la interfaz:

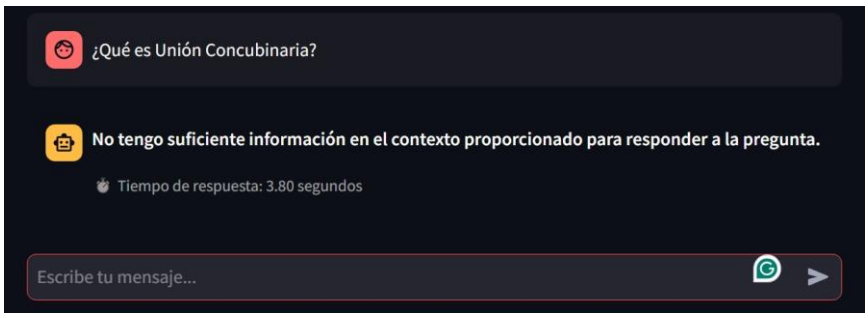
Interfaz funcionando:



Pregunta respondida correctamente y link del documento legal:



Respuesta a pregunta de una Ley no incluida entre los temas elegidos como fuente:



Backend

Para el backend, utilizamos el framework de FastAPI.

De manera muy resumida, lo que hicimos fue lo siguiente:

Carga de variables de entorno:

Cargamos variable de entorno (clave de API de OPENAI), creamos la app de FastAPI y añadimos CORS (para no tener problemas de seguridad).

System prompt y LLM

Cargamos el system prompt que creamos en un archivo Markdown dentro del directorio y cargamos el modelo con el método ChatOpenAI de Langchain.

Modelado de datos (input)

Creamos una clase con el modelado de datos que queremos, en este caso solamente el prompt. Esta clase hereda de BaseModel, un modelo de Pydantic.

Context Prompt

Luego se crea el context prompt, en el que se pasa al modelo el system prompt y el contexto.

Saludos y agradecimientos

Para esto, cargamos el modelo de Sentence Transformers, y pusimos dentro de una lista palabras como “hola”, “buen día”, entre otras. Lo mismo para los agradecimientos.

Después de eso, le dijimos al modelo que calcule los embeddings en estas listas. Para después, poner en una función que si la similaridad semántica del input es mayor a 0.6, pase como contexto strings vacíos (ponemos esto más tarde en el endpoint /chat).

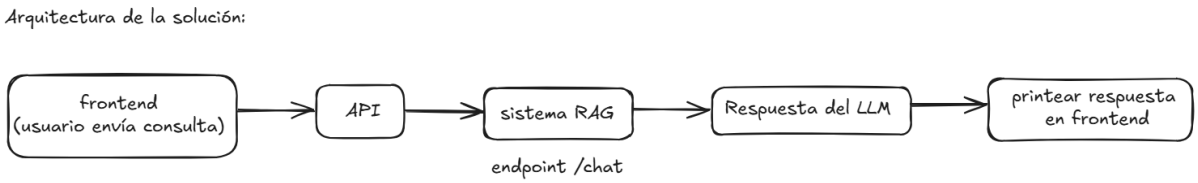
Endpoint principal “/chat”

Este endpoint hace la interacción entre el usuario y el chatbot. Entonces lo que hace este endpoint es un resumen de todo lo visto anteriormente. Recupera la información (con su link), construye el context prompt, extrae el link del context y luego responde la pregunta del usuario en streaming.

¿Por qué en streaming? Para mejorar los tiempos de respuesta. Al pedirle las respuestas en streaming el chatbot va a ir respondiendo a medida que vaya generando la respuesta, y no generar toda la respuesta y después escribirla. De alguna manera, va haciendo las dos tareas al mismo tiempo: escribir la respuesta y generarla.

También importante a destacar en esta parte es que se usan funciones asincrónicas, lo que quiere decir que no se ejecuta de forma secuencial, sino que permite pausar su ejecución mientras espera una operación lenta, y mientras tanto, puede ir ejecutando otros bloques de código.

Arquitectura de nuestra solución:



Conclusiones

Como conclusión, podemos afirmar que estamos satisfechos con los resultados obtenidos del Asistente Legal Uruguay, debido a la precisión de sus respuestas.

Como parte del proceso de aprendizaje, realizamos los siguientes cambios significativos:

- Sustituimos el modelo de embeddings de Hugging Face por el de OpenAI, y reemplazamos la base de datos vectorial ChromaDB por FAISS.
- Ajustamos el sistema para que realice *copy-paste* de los textos legales, dado que, en el ámbito jurídico, una misma idea puede tener múltiples interpretaciones dependiendo de cómo esté redactada.
- Al principio, al intentar implementar la feature de incluir el link, cuando el usuario saludaba al modelo, respondía con un link. Ahí fue cuando implementamos el modelo de

Sentence Transformer y el chatbot empezó a responder bien cuando se lo saludaba o agradecía.
- Pasamos de tener como fuente archivos de Word a links de IMPO.

Las oportunidades de mejora del chatbot de Derecho Uruguayo podrían ser las siguientes:

- Incorporación de memoria conversacional: Implementar mecanismos que permitan al chatbot recordar el historial de la conversación, para brindar respuestas más coherentes y contextualizadas a lo largo de la interacción. (Implementación de memoria)
- Uso de agentes de inteligencia artificial: Integrar agentes de AI que puedan coordinar múltiples herramientas o flujos de razonamiento, lo cual permitiría resolver tareas más complejas o realizar acciones específicas dentro del dominio jurídico.
- Evolución del prototipo hacia una solución más robusta: Actualmente, el chatbot se presenta como una prueba de concepto (POC). Se podría escalar a una aplicación más sólida, con un backend optimizado y un frontend más atractivo y profesional, por ejemplo sustituyendo el streamlit por un código en React.js, mejorando así la experiencia del usuario y su adopción en entornos reales.
- Ampliación de las fuentes legales: Incluir más fuentes relevantes (como doctrina, jurisprudencia o reglamentos administrativos) para enriquecer la base de conocimientos del modelo RAG.
- Mejor gestión de actualizaciones normativas: Diseñar un mecanismo automatizado para mantener actualizada la información legal frente a cambios en leyes, decretos o sentencias relevantes.
- Soporte multilingüe: Incorporar la capacidad de interactuar en varios idiomas si se amplía a públicos diversos.
- Autenticación y personalización: Permitir que el chatbot identifique usuarios recurrentes, ofrezca historial de consultas, o adapte sus respuestas a distintos perfiles, por ejemplo, estudiantes, abogados y ciudadanos. Implementación de login, mediante uso de bases de datos relacionales (SQL, MongoDB, u otras).