MODELOS DE LENGUAJE 2025 – TRABAJO PRACTICO

ALVARO CASTRO – AGUSTIN HEBERLING – FRANCISCO PEREZ

1- **Tema elegido para el asistente conversacional.**

Se optó por elegir temas referentes a la normativa de regulación y control del sistema financiero uruguayo, en particular la Circular No. 2473 emitida por el Banco Central del Uruguay (BCU).

2- **Justificación de la elección del tema, relevancia e interés.**

La circular en cuestión es utilizada a diario en el área de consultoría empresarial y auditoría externa por lo que es un documento de relevancia. Es de relevancia la obtención de la información sin tener que recorrer el documento o tenerlo memorizado. Como ejemplo, al no encontrar la información buscada dentro del documento se recurrió directamente al BCU quien contesto que la información solicitada se encontraba en un artículo especifico de la circular.

3- **Fuentes utilizadas para construir la base de conocimiento.**

Se obtuvo la circular directamente de la página del BCU: https://www.bcu.gub.uy/Acerca-de-BCU/Normativa/Documents/Reordenamiento%20de%20la%20Recopilación/Sistema%20Financiero/RNRCSF.pdf

4- **Descripción del pre procesamiento y parseo de los datos, incluyendo decisiones tomadas para su estructuración.**

El pre procesamiento y parseo principal del contenido del PDF se logra mediante dos funciones que implementamos. Las describimos a continuación:

a. limpiar_encabezados_pies

Elimina elementos repetitivos e irrelevantes del texto que provienen de la maquetación del PDF (como encabezados de página, pies o referencias legales estándar), que podrían contaminar los embeddings o confundir al modelo LLM.

Se usan expresiones regulares para detectar patrones específicos (referencias a “Última circular”, resoluciones con fechas, o “Antecedentes del artículo”). Se ejecuta un re.sub() para remover estos bloques de texto, ignorando mayúsculas/minúsculas (re.IGNORECASE) y se limpian líneas vacías consecutivas para mantener una estructura uniforme.

b. extraer_articulos_con_titulos_capitulos

Se parsea la estructura jerárquica del documento legal: Libro → Título → Capítulo → Sección → Artículo, y se extraen los artículos como bloques independientes con sus respectivos metadatos estructurales.

Se lee el PDF completo, página por página extrayendo el texto con PyMuPDF y se usa .get_text() para obtener contenido plano. Se invoca limpiar_encabezados_pies() para depurar el contenido antes del análisis estructural. Se identifican de encabezados jerárquicos mediante una expresión regular compuesta (encabezado_regex = r'^(LIBRO|T[ÍI]TULO|CAP[ÍI]TULO|SECCI[ÓO]N|ART[ÍI]CULO)\s+[^\n]*') detectando los bloques de libro, titulo, capitulo, sección y articulo asignando jerárquicamente una lógica de acumulación (si detecta LIBRO, actualiza la variable libro, si detecta TÍTULO, actualiza titulo, etc.) Finalmente, separa en fragmentos, ya que para cada artículo detectado se extrae el texto entre ese encabezado y el siguiente y se guarda un diccionario con {"libro": ..., "titulo": ..., "capitulo": ..., "seccion": ..., "articulo": ..., "contenido": ...}.

Como resultado se obtienen una lista de artículos individuales, cada uno enriquecido con su contexto jerárquico, lo que mejora la calidad semántica del embedding, la trazabilidad legal y el formato de respuesta del LLM.

5- **Modelo LLM utilizado, framework y método de ejecución (por ejemplo, Ollama, Transformers, Llama.cpp, etc.)**

Se utiliza un modelo Llama 3.2 (meta-llama/Llama-3.2-3B-Instruct) de 3 mil millones de parámetros del Hugging Face, cuyo propósito es responder preguntas conversacionales basadas en un contexto, ideal para tareas tipo RAG.

Entendemos que Llama 3.2 Instruct es adecuado pues es un modelo optimizado para seguir instrucciones en tareas conversacionales, ideal para preguntas legales tipo asistente jurídico. Su tamaño moderado (3B) balancea bien entre capacidad de comprensión y velocidad de inferencia (menor costo computacional).

Como framework y método de ejecución se utilizan transformers de Hugging Face directamente en el código mediante la función pipeline() de transformers, con device_map para GPU si está disponible y torch_dtype=”auto” para que el sistema decida. Se realiza posteriormente su integración con LangChain con HuggingFacePipeline para convertir el pipeline de transformers en un LLM usable dentro del flujo RAG de LangChain.

La ejecución vía Transformers permite máximo control sobre la generación (tokens, temperatura, etc.), la integración directa con pipelines Python y LangChain y evitar depender de servicios externos como OpenAI o Ollama.

6- **Sistema de embeddings implementado, indicando el modelo utilizado, justificación de su elección y configuración del índice vectorial.**

Se implementó el modelo intfloat/multilingual-e5-base de Hugging Face para la generación de embeddings semánticos a partir del contenido del documento PDF. Este modelo está diseñado para tareas de recuperación de información (retrieval) y

generación de representaciones vectoriales densas que capturan el significado de frases o párrafos completos.

Nos decantamos por este modelo debido a que es un modelo multilingüe, lo cual lo hace adecuado para textos legales en español. Hay un balance entre rendimiento y eficiencia, pues a diferencia de su versión "large", el modelo "base" ofrece un equilibrio adecuado entre precisión semántica y uso de recursos computacionales, especialmente importante al trabajar con CPUs y entornos locales limitados, como es nuestro caso. Finalmente, ofrece compatibilidad con tareas de RAG (Retrieval-Augmented Generation) siendo la arquitectura del modelo optimizada para contextos en los que se desea buscar información en bases vectoriales antes de generar una respuesta con un modelo LLM.

Cabe mencionar que utilizamos originalmente un modelo multilingual-e5-large que daba excelentes resultados en el notebook original en Google colab. Los tiempos de procesamiento en CPU del PC nos llevaron a probar el modelo paraphrase-multilingual-MiniLM-L12-v2 (mucho más liviano, tiene buen rendimiento en español, muy rápido para generar embeddings en CPU, compatible con LangChain). Este modelo daba en el notebook resultados razonables, pero con detalles. Finalmente nos decantamos por el modelo e5-base por su buen balance entre precisión y velocidad.

Para la configuración del índice vectorial, utilizamos la librería FAISS (Facebook AI Similarity Search) para construir un índice vectorial que permite realizar búsquedas semánticas eficientes. El índice se construye sobre los embeddings generados y permite encontrar los pasajes más relevantes del documento en función de la similitud con la consulta del usuario.

Tenemos una configuración para un tipo de índice con FAISS plano con búsqueda por similitud coseno o dot-product, dependiendo de la normalización de los vectores. Se almacena el índice para evitar re-embeddings cada vez que se ejecuta la aplicación y se usó FAISS.from_documents(...) integrando directamente con LangChain, lo que facilita su uso dentro del sistema RAG.

7- Diseño del sistema RAG: arquitectura general, flujo de recuperación y generación, y prompting aplicado.

El sistema implementado sigue el enfoque de RAG (Retrieval-Augmented Generation), una arquitectura que combina técnicas de búsqueda semántica y generación de lenguaje natural para responder preguntas basadas en nuestro documento legal en formato PDF. Este enfoque permite que un modelo generativo (LLaMA) produzca respuestas contextualizadas, informadas por pasajes relevantes del documento.

El sistema se compone de tres módulos principales:

a) Pre procesamiento y embedding del documento:

- El PDF se divide en fragmentos textuales (chunks).
- Cada fragmento se vectoriza usando el modelo intfloat/multilingual-e5-base.
- Los vectores se almacenan en un índice de FAISS, que actúa como base vectorial para recuperación semántica.

b) Flujo de Recuperación de información (Retrieval):
- Cuando el usuario envía una pregunta en lenguaje natural desde la interfaz (Streamlit), esta se convierte en un vector usando el mismo modelo de embeddings.
- Se realiza una búsqueda de similitud en el índice FAISS para recuperar los fragmentos del documento más relevantes (tipicamente los k más similares).
- Estos fragmentos representan el contexto que se entregará al modelo generativo.

c) Flujo de generación (Generation):
- Los fragmentos recuperados se insertan en una plantilla de prompt (prompting) junto con la pregunta del usuario.
- Este prompt enriquecido se envía al modelo de lenguaje meta-llama/Llama-3.2-3B-Instruct mediante Hugging Face Transformers.
- El modelo produce una respuesta en lenguaje natural que intenta responder la pregunta utilizando el contenido extraído del documento.

El sistema usa un prompt estructurado con la siguiente forma:

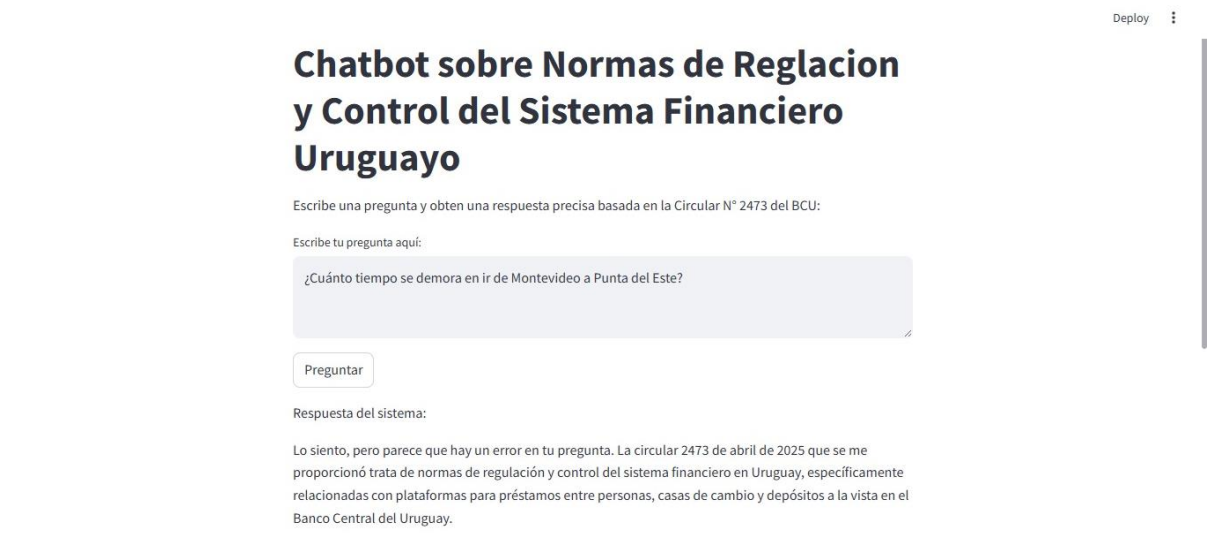
- Contexto: <fragmentos recuperados del documento>
- Pregunta: <pregunta del usuario>
- Respuesta: <respuesta dada por el sistema>

Este tipo de prompting guía al modelo a generar una respuesta exclusivamente basada en el contenido disponible, minimizando la alucinación y alineando la respuesta con el contexto legal proporcionado.

Entendemos que con este diseño logramos:

a) Eficiencia y precisión: al separar la recuperación del conocimiento (via FAISS) de la generación de texto (via LLaMA), el sistema responde con información precisa y actual sin necesidad de entrenar el modelo de lenguaje.
b) Escalabilidad: la arquitectura RAG permite incorporar más documentos o normas simplemente añadiendo nuevos embeddings al índice FAISS.
c) Adaptabilidad lingüística: tanto el modelo de embeddings como el generativo están adaptados al español (vía modelos multilingües), lo que es esencial para el dominio jurídico uruguayo.

8- Capturas de pantalla del sistema funcionando desde la interfaz de Streamlit.



9- Resumen del funcionamiento del backend, explicando brevemente cómo se conectan los distintos componentes.

El backend está desarrollado en FastAPI y actúa como intermediario entre la interfaz web (Streamlit) y los modelos de procesamiento. Su función es coordinar el flujo de datos desde la pregunta del usuario hasta la generación de una respuesta informada por el contenido del documento PDF. Los distintos componentes se conectan de la siguiente manera:

a) Recepción de la pregunta: el backend expone un endpoint /ask/ que recibe una pregunta enviada por el usuario desde Streamlit (mediante una solicitud HTTP POST).
b) Vectorización de la pregunta: se utiliza el modelo intfloat/multilingual-e5-base de Hugging Face para convertir la pregunta en un vector de embeddings.

c) Recuperación semántica (FAISS): el vector de la pregunta se compara contra el índice FAISS, donde están almacenados los vectores de fragmentos del documento PDF. Se recuperan los k fragmentos más relevantes, que forman el contexto para la respuesta.
d) Construcción del prompt: se crea un prompt que incluye el contexto y la pregunta del usuario, siguiendo una estructura predefinida.
e) Generación de la respuesta: el prompt se pasa al modelo meta-llama/Llama-3.2-3B-Instruct mediante la librería transformers de Hugging Face. Se genera una respuesta en lenguaje natural basada exclusivamente en el contexto proporcionado.
f) Respuesta al frontend: la respuesta generada se devuelve al frontend en formato JSON. Streamlit muestra la respuesta al usuario en la interfaz web.

En definitiva, el frontend (Streamlit) envía pregunta vía requests.post() y presenta la respuesta al usuario; FastAPI (Backend) procesa, recupera contexto y genera respuesta, respondiendo con la respuesta generada; FAISS y Hugging Face proveen los mecanismos de recuperación y generación.

Este diseño modular y asincrónico permite mantener una clara separación de responsabilidades, optimizando el mantenimiento y la escalabilidad del sistema.

10- Conclusiones del trabajo, incluyendo aprendizajes, dificultades, oportunidades de mejora y próximos pasos.

a) Aprendizajes:

Se comprendió y aplicó el funcionamiento de un sistema RAG (Retrieval-Augmented Generation), integrando recuperación semántica con generación de texto en lenguaje natural.

Se adquirió experiencia práctica en el uso de modelos de embeddings multilingües como intfloat/multilingual-e5-base, balanceando rendimiento y eficiencia.

Se profundizó en herramientas del ecosistema de NLP moderno, como Hugging Face, FAISS, LangChain y FastAPI (también vistas en materias como Deep learning y Analítica Big Data).

Se reforzaron conceptos de despliegue y ejecución de APIs y aplicaciones en distintos entornos, incluyendo la gestión de entornos virtuales, puertos, y autenticación segura de tokens.

Se logró dockerizar un proyecto complejo, con dependencias pesadas y múltiples servicios corriendo simultáneamente. Esto es un paso clave hacia la portabilidad y reproducibilidad.

b) Dificultades encontradas

El uso inicial del modelo e5-large provocó problemas de memoria y rendimiento que comprometían la usabilidad del sistema, especialmente en nuestro equipo sin GPU.

La configuración de la conexión entre frontend y backend requirió solución de errores comunes, como puertos bloqueados, y sincronización de servicios.

La autenticación con Hugging Face presentó desafíos en términos de seguridad del token y compatibilidad con Windows, resolviéndose mediante entrada interactiva en consola.

La construcción de la imagen con Docker fue muy lenta debido a un contexto muy grande y requisitos muy pesados en requirements que requirió el uso de dockerignore. También demandó modificar la lógica de la app para la autentificación en Hugging Face. Finalmente, no se logró ver funcionando el sistema dockerizado.

c) Oportunidades de mejora:

Incorporar persistencia en disco del índice FAISS para evitar recomputaciones innecesarias al reiniciar la app.

Implementar un sistema de pre procesamiento de documentos más robusto, con limpieza de texto, manejo de tablas y reconocimiento de secciones clave.

Explorar el uso de modelos cuantizados o versiones optimizadas (gguf, llama.cpp) que permitan usar modelos más grandes con menos recursos.

Añadir logs detallados y manejo de errores en frontend y backend para mejorar la experiencia de usuario y la depuración, así como mensajes de tiempo de ejecución.

Optimizar imagen Docker mediante el uso de multi-stage builds para minimizar el tamaño final y considerar congelar dependencias (pip freeze > requirements.txt) para evitar builds variables.

d) Próximos pasos:

Evaluar la posibilidad de ejecutar el sistema en un entorno de producción en la nube (ej. Render, Hugging Face Spaces, o EC2), eliminando restricciones locales.

Extender el sistema para permitir carga dinámica de documentos, generando un índice FAISS por documento o por usuario.

Explorar el uso de retrieval híbrido (texto + metadata), permitiendo búsquedas más refinadas.

Añadir funcionalidades como descarga de respuestas, historial de preguntas, o feedback del usuario para mejorar el sistema iterativamente.