Trabajo Práctico Final

“Instituto Politécnico Formosa”

Tema: Implementación de un Sistema RAG usando LangChain y Ollama

Materia: S.A. III - Modelos y Aplicaciones de la IA

Profesor:

- Bogarin Sebastian

Integrantes del grupo:

- Acosta Gabriel
- Acuña Melanie
- Mazza Agustin
- Romero Ezequiel

2024

Índice

Introducción

El presente informe tiene como objetivo exponer el propósito y la funcionalidad del sistema RAG (Retrieval-Augmented Generation), diseñado para facilitar la recuperación de información precisa y relevante en documentos específicos. Este sistema incluye una interfaz intuitiva de usuario y un backend robusto, los cuales se comunican eficientemente a través de LangChain mediante la API de Ollama.

El sistema RAG responde a la necesidad de mejorar la precisión y accesibilidad de la información en bases de datos documentales, particularmente en aquellos casos donde el volumen de datos es elevado o los documentos contienen contenido especializado. Este enfoque permite a los usuarios encontrar rápidamente respuestas directas a sus consultas, utilizando técnicas de recuperación y generación de datos avanzadas. Además, la integración de LangChain potencia el sistema al gestionar la interacción en lenguaje natural y al acceder a información contenida en documentos específicos, optimizando el tiempo y la calidad de las respuestas obtenidas.

Descripción del proyecto

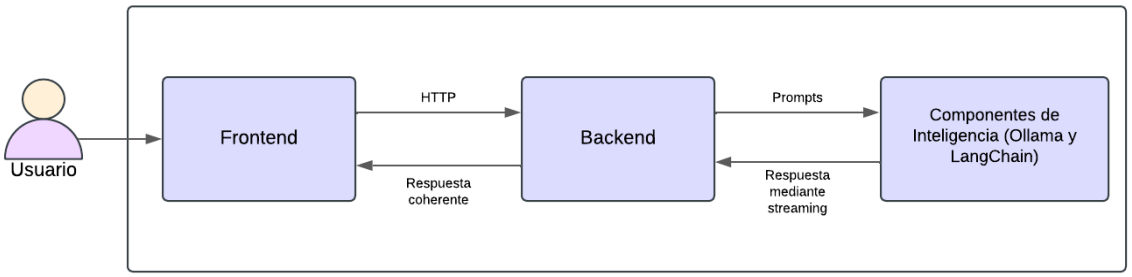
Este proyecto consiste en el desarrollo de un chatbot con capacidades avanzadas de Recuperación Aumentada Generativa (RAG) para responder preguntas específicas sobre documentos PDF cargados por el usuario. El sistema, basado en tecnologías de LangChain y Ollama, utiliza técnicas de recuperación de información y generación de respuestas para ofrecer respuestas precisas y contextualizadas. Su objetivo es brindar al usuario una interacción ágil y efectiva para obtener datos exactos del contenido cargado, logrando así aprovechar al máximo la información contenida en cada documento.

El funcionamiento principal del sistema inicia cuando el usuario carga el documento PDF que desea consultar. Este archivo se convierte en la fuente de datos principal, permitiendo al chatbot procesar su contenido y extraer información relevante al responder preguntas. Cada vez que el usuario cargue un nuevo documento, éste se mostrará automáticamente en un menú desplegable (select) en la parte superior de la pantalla, junto a otros documentos que el usuario haya cargado anteriormente. De esta manera, el sistema facilita la gestión y selección de archivos, asegurando un acceso rápido a las fuentes de información para una experiencia de consulta optimizada.

Alcances del proyecto

- Carga y Gestión de Documentos: El usuario puede cargar múltiples documentos PDF, los cuales se almacenarán y estarán disponibles para futuras consultas en un menú select, facilitando el acceso rápido.
- Interacción Conversacional con el Chatbot: El chatbot permitirá realizar preguntas relacionadas específicamente con el contenido del PDF seleccionado, generando respuestas relevantes a partir de la información contenida en el archivo. Esto abarca preguntas de detalle y contexto del documento.
- Recuperación Aumentada Generativa: A través de la integración de LangChain y Ollama, el sistema combina técnicas avanzadas de recuperación de datos y generación de respuestas para asegurar que el usuario obtenga respuestas precisas y contextualizadas.
- Interfaz de Usuario Intuitiva: El usuario interactúa con una interfaz que muestra sus documentos disponibles, asegurando una navegación fluida y un acceso directo al documento de interés para formular preguntas específicas.

Arquitectura del sistema



Frontend

- Funcionalidad principal:

El frontend, desarrollado con React, ofrece una interfaz gráfica para los usuarios. Permite cargar documentos PDF, seleccionar un documento, realizar consultas y visualizar respuestas.

- Interacción con Backend:

Las consultas y solicitudes de carga de documentos se envían al backend a través de endpoints REST utilizando HTTP.

Backend

- Funcionalidad principal:
    - El backend, construido con Python y FastAPI, maneja la lógica de negocio del sistema.
    - Procesa los documentos PDF cargados, dividiéndolos en fragmentos manejables.
    - Genera embeddings y recupera los fragmentos relevantes para las consultas de los usuarios.
    - Envía prompts al modelo LLama y devuelve las respuestas al frontend.
- Endpoints principales:
    - /upload-pdf/: Para cargar documentos PDF.
    - /ask/: Para procesar preguntas y generar respuestas en tiempo real.
    - /list-pdfs/: Para listar los documentos cargados.
    - /process-pdf/: Para dividir y procesar documentos cargados.

Componentes de Inteligencia (Ollama y LangChain)

- Ollama:
    - Maneja la interacción con el modelo de lenguaje LLaMA (llama3.2:1b).
    - Genera respuestas basadas en el contexto proporcionado por el backend.
- LangChain:
    - Facilita la gestión de prompts y flujos de generación de lenguaje natural.
    - Convierte consultas en prompts estructurados y se comunica con Ollama.
- Embeddings y recuperación:
    - Se generan representaciones numéricas del contenido del documento mediante un modelo de embeddings (FastEmbedEmbeddings).
    - Permite comparar la consulta del usuario con los fragmentos del documento para identificar los más relevantes.

Configuración de entorno

Tecnologías utilizadas en el servidor:

- Python
- Landchain
- FastApi
- Ollama y de modelo llama3.2:1b

Tecnologías utilizadas en el cliente:

- Tailwind
- React
- React icons
- React-dom
- React Toastify

Sigue estos pasos para instalar las dependencias y ejecutar el proyecto:

- Clona este repositorio en tu máquina local:

git clone https://github.com/gabykap29/proyecto_final_chatbot.git

- Navega al directorio del proyecto:

cd proyecto_final_chatbot/server

- Crea un entorno virtual (opcional pero recomendado):

python -m venv venv
source venv/bin/activate # En Windows usa `venv\Scripts\activate`

- Instala las dependencias:

pip install -r requirements.txt

- Ejecuta el servidor:

uvicorn main:app --reload

- Ingresa a la carpeta del cliente

cd proyecto_final_chatbot/client

- Configura la variable de entorno con la url base del servidor de Python.

VITE_API_SERVER=

- Instala todas las dependencias necesarias

npm install

- Ejecuta el servidor

npm run dev

Diseño y Desarrollo del Frontend

- Lenguaje y Framework:
    - **Lenguaje:** JavaScript, específicamente con React
    - **Framework de desarrollo frontend:** React
    - **Herramienta de compilación:** Vite

- Flujo de Usuario:

La interfaz de usuario permite una interacción sencilla y directa para realizar consultas y visualizar respuestas sobre un documento cargado. Al iniciar, el usuario puede cargar un archivo PDF desde su dispositivo, el cual se mostrará en un menú desplegable junto con otros documentos que haya subido previamente. Una vez seleccionado el documento, el usuario puede escribir una consulta específica sobre su contenido en el campo de preguntas. Al enviar la consulta, el sistema procesa la solicitud y muestra la respuesta en pantalla, proporcionando una respuesta contextualizada basada en la información del archivo. Este flujo asegura que el usuario pueda navegar entre documentos, hacer consultas rápidas y recibir respuestas precisas en una experiencia de uso fluida y eficiente.

- Implementación del Código:
    - Componente

El **componente Chat** se encarga de gestionar la interacción entre el usuario y el chatbot. Utiliza el estado messages para almacenar una lista de mensajes, cada uno con el texto y el remitente (user o bot), y el estado input para capturar el valor actual en el campo de entrada. La función handleSend agrega el mensaje del usuario a la lista de mensajes y luego limpia el campo de entrada. Al renderizar, se utiliza map para mostrar cada mensaje alineado a la derecha o izquierda según el remitente, y los mensajes del chatbot incluyen un ícono que representa al bot, proporcionando así una experiencia visual clara y organizada.

- Vistas

MainView (contenida en App.jsx):
La vista principal muestra el título, el componente de chat, y el área de entrada. Aquí el usuario puede realizar todas las acciones: seleccionar un documento, escribir una consulta y ver la respuesta del chatbot en el mismo lugar.

Desarrollo del backend

- Lenguaje y Framework:
  - **Lenguaje**: Python con LangChain como herramienta para integrar y coordinar flujos de trabajo con modelos de lenguaje.
  - **Framework:** FastAPI: para la construcción de un servidor web con python.
  - **Ollama:** Como plataforma de gestión de modelos de lenguajes open source.
  - **Modelo**: Ollama con el modelo LLaMA 3.2:1b

- Endpoints de la API:
  1. **/ask/ (POST):** Este endpoint recibe una pregunta en formato de texto (como parámetro) y utiliza el servicio QuestionAnsweringService para generar una respuesta en tiempo real, basada en el contenido de los documentos procesados previamente.
  2. **/upload-pdf/ (POST)**: Este endpoint permite a los usuarios cargar archivos PDF al sistema. Una vez recibido el archivo, se guarda en una carpeta específica del servidor y se almacena su ruta.
  3. **/list-pdfs (GET)**: Este endpoint devuelve una lista de todos los archivos PDF disponibles en el servidor. Permite al usuario consultar los archivos que han sido cargados previamente.

4. **/process_pdf (POST)**: Este endpoint procesa un archivo PDF específico, tomando el nombre del archivo como parámetro, y realiza el análisis de contenido utilizando el servicio QuestionAnsweringService.

- Integración con Ollama y LangChain:
    - **Configuración del modelo**: El código usa LangChain para conectar con Ollama, especificando el modelo LLaMA (llama3.2:1b) y un servidor de Ollama al que se hacen las peticiones. Esto se configura en la clase QuestionAnsweringService, que define cómo el modelo interactúa con el sistema.
    - **Generación de respuestas**: Cuando el usuario hace una pregunta, el sistema recoge el contexto de un documento PDF cargado previamente y lo combina con la pregunta del usuario. LangChain luego crea un prompt que se envía a Ollama. Este prompt le indica al modelo que utilice el contexto del PDF para generar una respuesta.
    - **Interacción con el modelo**: Usando el método stream() de LandChain, el sistema envía la pregunta al modelo y recibe la respuesta en fragmentos, lo que mejora la experiencia al proporcionar respuestas más rápidamente.
    - **Uso de embeddings**: Además, el sistema utiliza un modelo de embeddings para transformar el texto del PDF en representaciones vectoriales, lo que ayuda a mejorar la precisión y la relevancia de las respuestas.

- Documentado del código (partes relevantes):

Clase QuestionAnsweringService:

Función: Esta clase maneja la lógica de la respuesta a las preguntas. Inicializa el modelo LLaMA a través de Ollama y también maneja la carga y procesamiento de documentos PDF. Los documentos PDF se dividen en "chunks" de texto, y estos fragmentos se utilizan para generar respuestas relevantes a las preguntas del usuario. Ademas, utilizamos el parámetro “base_url” con la url de un túnel montado con ngrok, con el fin de utilizar el servicio de ollama montado en otro equipo.

```python
class QuestionAnsweringService:
    def __init__(self, model_name="llama3.2:1b"):
        self.llm = ChatOllama(model=model_name, base_url="https://4dca-138-121-113-25.ngrok-free.app",streaming=True)
        self.chunks = []  # Inicializar una lista vacía para los chunks

        # Modelo de embeddings
        self.embed_model = FastEmbedEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")

        # Definir prompt template
        custom_prompt_template = """Usa la siguiente información para responder a la pregunta del usuario.
        Si la respuesta no se encuentra en dicha información, di que no sabes la respuesta.

        Contexto: {context}
        Pregunta: {question}

        Solo devuelve la respuesta útil a continuación y nada más. Responde siempre en español:
        """
        self.prompt = PromptTemplate(
            template=custom_prompt_template,
            input_variables=['context', 'question']
        )
```

Método process_pdf:

Función: Este método procesa un archivo PDF y lo divide en fragmentos de texto que luego se pueden usar para generar respuestas a las preguntas del usuario.

```python
    def process_pdf(self, file_path: str):
        # Verificar que el archivo exista
        if not os.path.exists(file_path):
            raise FileNotFoundError(f"El archivo {file_path} no existe.")

        # Usar PyMuPDFLoader para cargar el PDF desde el archivo
        loader = PyMuPDFLoader(file_path)
        data_pdf = loader.load()

        # Dividir el PDF en chunks
        text_splitter = RecursiveCharacterTextSplitter(chunk_size=2000, chunk_overlap=500)
        self.chunks = text_splitter.split_documents(data_pdf)
```

Método stream_answer:

Función: Este método utiliza los fragmentos de texto procesados del PDF para generar una respuesta a la pregunta del usuario, enviando el contexto y la pregunta al modelo LLaMA. La respuesta se devuelve en fragmentos, lo que permite una experiencia más rápida y fluida.

```python
    def stream_answer(self, question: str):
        context = "\n".join([chunk.page_content for chunk in self.chunks])
        formatted_prompt = self.prompt.format(context=context, question=question)

        messages = [
            {"role": "system", "content": "Eres un asistente de inteligencia artificial. SI recibes un insulto o palabras como 'inutil' o 'no sir
            {"role": "user", "content": formatted_prompt},
        ]

        response = self.llm.stream(messages)

        # Usar yield para retornar los fragmentos de la respuesta en streaming
        for chunk in response:
            yield chunk.content

```

- Embeddings y Recuperación de Información:

1. Carga y Preprocesamiento del Documento (PDF)

Cuando se sube un PDF, el sistema extrae el texto y lo divide en fragmentos más pequeños para evitar sobrecargar el modelo con documentos largos. Esto se hace usando herramientas que segmentan el texto en partes manejables, manteniendo la eficiencia y sin perder detalles importantes.

2. Embeddings (Generación de Representaciones Numéricas)

Cada fragmento de texto se convierte en una representación numérica (embedding) utilizando un modelo especializado. Estos embeddings son vectores que representan el significado del texto, permitiendo que el sistema identifique rápidamente fragmentos relevantes y los compare con la consulta del usuario.

3. Recuperación de Información

Cuando un usuario pregunta algo, el sistema convierte la consulta en un embedding y lo compara con los embeddings de los fragmentos del documento. Los fragmentos más relevantes se seleccionan para generar una respuesta.

4. Generación de Respuestas con el Modelo de Lenguaje (LLaMA)

Los fragmentos relevantes se envían al modelo LLaMA, que usa el contexto y la pregunta para generar una respuesta coherente en tiempo real, que luego se envía al usuario mediante streaming.

Conclusión

La capacidad del sistema para procesar documentos extensos mediante fragmentación, generación de embeddings, y recuperación relevante, lo posiciona como una herramienta poderosa en ámbitos donde la consulta documental especializada es crucial, como la investigación, educación y análisis corporativo. Además, el diseño extensible y modular asegura su adaptabilidad a diferentes necesidades y escenarios futuros.

En resumen, este proyecto demuestra cómo la inteligencia artificial puede integrarse de manera efectiva en sistemas de gestión documental, mejorando la accesibilidad y usabilidad de la información mediante un enfoque práctico y orientado al usuario.