Mini chatbot (Promtior)

Objetivo

El objetivo de este mini Proyecto es desarrollar un chatbot funcional utilizando LLM para contestar preguntas relacionadas al siguiente contenido ingerido:

1. Página web de Promtior (https://promtior.ai/, https://promtior.ai/service/, https://promtior.ai/contacto/ y https://promtior.ai/user-cases/)
2. El PDF adjunto en esta documentación (AI Engineer.pdf).

Funcionamiento lógico

Para desarrollar el chatbot se utilizaron 3 componentes claves.

- **OpenWebUI**: Este componente provee una interfaz web estándar para poder interactuar el con el chatbot, al mismo tiempo también se encarga de almacenar el historial de los chats.

- **FastAPI**: Esta api se encarga de lo siguiente:
    - Exponer endpoints que siguen el estándar de OllamaAPI (solo los básicos necesarios para este proyecto), ya que el componente anterior se comunica con este estándar y no es posible modificarlo.
    - Procesar los datos al inicio y cargarlos en una base de datos vectorial simple (FAISS, aunque se podría haber usado chromaDB).
    - Tomar la solicitud de chat, buscar en la base de datos vectorial los "documentos/chunks" relacionados a la consulta y luego enviarlos a la API de Ollama.
- **OllamaAPI**: esta api permite interactuar con el modelo LLM/SLM (phi3:mini) para generar la respuesta en base a la consulta y los datos obtenidos de la base vectorial.

Funcionalidades de la api:

Se intento implementar el patron que permite contextualizar al modelo con el historial del chat (ejemplo provisto por langchain), el mismo funcionaba, pero retornaba ambas respuestas del modelo (la del contexto y la respuesta final), por lo que se definió que no sea “history-aware”. El código sigue implementado, pero se utiliza solo la chain final.

Si bien el frontend envía el historial entero del chat, solo se toma el último mensaje.

La fast api actualmente está implementada con el método async stream del LLM para poder retornar un streaming de datos y no tener que esperar a la respuesta completa del modelo. Esto le da sensación de agilidad al sistema.

Endpoints

Host OpenWebUI: http://raikou.ddns.net:3000/
Host FastAPI: http://raikou.ddns.net:8000/

Path	Method	Body	Description
/agent	Post	{input:string}	Endpoint antiguo de la api, sirve para comprobar el funcionamiento básico del chat.
/api/version	Get		Devuelve la versión de ollamaAPI (necesario para el front)
/api/tags	Get		Devuelve los modelos de LLM disponible (Acutalmente solo 1)

/api/chat	POST	{
		stream:bool,
		model:string,
		options: object (por defecto vacio),
		Messages: Historial del chat
		}	

Deployment

Para el deployment se crea un simple Docker-compose donde se ejecutan los 3 componentes. (Ver Docker-compose.yml). Genera servicios básicos para que exista comunicación entre ellos.

Llegado el caso se podría haber creado un terraform sencillo que hiciera deploy de los diferentes componentes en alguno de los servicios de nube. Podría utilizar módulos para poder reutilizar el código en otros proyectos fácilmente.

Finalmente, no se terminó ejecutando ya que:

1. AWS Lambda no soporta GPU, por lo que el cálculo de vectores y la performance del LLM hubiera sido muy pobre
2. Crear un cluster ECS/EKS requiere de tiempo y configuración y no lo ameritaba para una simple demo del chat.
3. Costo

Finalmente se decide ejecutar el Docker-compose en una PC local, se exponen los puertos necesarios a internet y se utiliza DynamicDNS para que quede todo bajo un hostname y que no tengamos problema con la IP dinámica de Antel.

Otro plus grande es que se pudo aprovisionar una GPU, reduciendo los tiempos de respuesta del modelo bastante.

En el siguiente diagrama vemos como esta deployado: