Introducción

En el marco de la prueba técnica "Promptior Chatbot", se requirió el desarrollo de un chatbot capaz de responder preguntas específicas sobre la empresa Promptior utilizando una arquitectura RAG (Retrieval-Augmented Generation). Esta arquitectura combina la recuperación de documentos con la generación de respuestas a partir de modelos de lenguaje de gran tamaño (LLMs).

Para la implementación se utilizó el framework LangChain, que ofrece herramientas especializadas para el desarrollo de aplicaciones basadas en LLMs, facilitando procesos como la carga de documentos, la indexación y la generación de respuestas. En particular, se utilizó LangServe para exponer el modelo como una API REST, permitiendo la consulta de información mediante peticiones HTTP.

El chatbot se desarrolló con GPT-3.5-turbo de OpenAI como modelo de generación de lenguaje, y ChromaDB como base de datos vectorial para el almacenamiento y recuperación de embeddings. Los documentos de referencia incluyen un archivo PDF y los contenidos extraídos de las páginas web principales de Promptior.

Durante el desarrollo, se incorporaron mecanismos de monitoreo mediante LangSmith, permitiendo el seguimiento de llamadas al modelo y optimizando costos en el uso de la API de OpenAI. Finalmente, la solución fue desplegada en AWS, utilizando la interfaz de usuario predeterminada de LangServe (Playground) en lugar del frontend dedicado que fue desarrollado para el despliegue local.

Link de aplicación AWS

Playground

Resumen del Proyecto

Enfoque de la Implementación

El enfoque adoptado para la prueba fue aprovechar al máximo las herramientas que ofrece LangChain y ajustar los procedimientos a la versión v0.2, la última que mantiene compatibilidad con LangServe. Se optó por una implementación modular que permitiera cargar y procesar documentos de manera eficiente, almacenando sus representaciones en una base de datos vectorial para su posterior recuperación.

Carga y Procesamiento de Datos

- Se empleó OpenAIEmbeddings para vectorizar los documentos.

- Se utilizaron PyPDFLoader y WebBaseLoader para cargar información desde un PDF y dos páginas web.
- Se segmentaron los textos mediante RecursiveCharacterTextSplitter con fragmentos de 1000 caracteres y un solapamiento de 100 caracteres.
- Los embeddings se almacenaron en ChromaDB.

Pipeline de Consulta y Respuesta

- Se definió un prompt estructurado con contexto extraído de la base vectorial.
- Se implementó un retriever con k=5, ajustado tras pruebas para mejorar la relevancia de las respuestas.
- El pipeline incluye validación de entrada, recuperación de contexto y generación de respuesta con GPT-3.5-turbo.
- La API fue expuesta mediante LangServe con soporte para CORS.

Desafíos y Soluciones

Uno de los principales desafíos fue encontrar el equilibrio adecuado en los parámetros de recuperación de información. Inicialmente, `k=2` resultó insuficiente, lo que llevó a un aumento a `k=5`. Sin embargo, tras nuevas pruebas, se determinó que `k=3` podría proporcionar respuestas más precisas y evitar la inclusión de información redundante.

Otro inconveniente importante fue el impacto en la latencia del sistema. Dado que el chatbot no implementa `streaming` en las respuestas, el usuario puede percibir una demora mayor a la real, ya que la generación del texto ocurre en su totalidad antes de ser enviada.

El uso de LangServe presentó complicaciones adicionales, ya que esta herramienta ha quedado obsoleta y no recibe actualizaciones. Integrarla en el proyecto requirió adaptar soluciones a una documentación desactualizada y resolver problemas derivados de la falta de soporte oficial. Si bien permitió exponer el modelo de manera rápida, en el futuro podría ser conveniente evaluar alternativas más modernas y mantenidas por la comunidad como lo es LangGraph.

Otro aspecto crítico fue la gestión de credenciales y datos sensibles. Al preparar el código para despliegue en un repositorio público y su eventual implementación en AWS, fue necesario adoptar buenas prácticas para el almacenamiento seguro de claves API. Se implementaron mecanismos como variables de entorno y archivos secretos montados en contenedores, evitando la exposición de credenciales en el código fuente. Este enfoque minimizó riesgos de seguridad y aseguró que las claves no fueran comprometidas en el proceso de desarrollo y despliegue.

Posibles Mejoras Futuras y Conclusión

- Ajuste dinámico de parámetros (k y modelo de lenguaje) para optimizar precisión y eficiencia.

- Historial de conversación para mejorar la coherencia en respuestas sucesivas.
- Despliegue de un frontend propio, en lugar de depender del Playground de LangServe.

La solución final proporciona un chatbot funcional y eficaz para la recuperación de información sobre Promptior, con un diseño modular que permite futuras mejoras y expansiones.

Debo destacar que la prueba técnica fue muy enriquecedora ya que me motivó a realizar un desarrollo y despliegue desde 0, algo que no había hecho desde hace mucho tiempo y que me permitió experimentar y poner a prueba los conocimientos que fui adquiriendo a lo largo de mi carrera profesional en materia de resolución de problemas, consulta de fuentes de información y formulación de soluciones adecuadas.

Introduction

As part of the technical test "Promptior Chatbot", the task was to develop a chatbot capable of answering specific questions about the company Promptior using a Retrieval-Augmented Generation (RAG) architecture. This architecture combines document retrieval with response generation based on large language models (LLMs).

For the implementation, the LangChain framework was used, as it provides specialized tools for developing LLM-based applications, simplifying processes such as document loading, indexing, and response generation. In particular, LangServe was used to expose the model as a REST API, allowing information retrieval via HTTP requests.

The chatbot was built using OpenAI's GPT-3.5-turbo as the language generation model and ChromaDB as the vector database for storing and retrieving embeddings. The reference documents included a PDF file and content extracted from Promptior's main web pages.

During development, monitoring mechanisms were integrated using LangSmith, enabling tracking of model calls and optimizing API costs. Finally, the solution was deployed on AWS, using the default LangServe Playground interface instead of the dedicated frontend that was developed for local deployment.

AWS Application

Playground

Project Summary

Implementation Approach

The approach taken for this test was to maximize the use of LangChain's tools while adapting the workflow to version v0.2, the last one compatible with LangServe. A modular implementation was chosen, allowing for efficient document loading and processing, with their vectorized representations stored in a vector database for later retrieval.

Data Loading and Processing

- OpenAIEmbeddings was used to vectorize documents.
- PyPDFLoader and WebBaseLoader were used to load information from a PDF and two web pages.
- Texts were split into segments using RecursiveCharacterTextSplitter, with chunks of 1,000 characters and an overlap of 100 characters.
- Embeddings were stored in ChromaDB.

Query and Response Pipeline

- A structured prompt was defined with context retrieved from the vector database.

- A retriever was implemented with k=5, fine-tuned after testing to improve response relevance.
- The pipeline includes input validation, context retrieval, and response generation using GPT-3.5-turbo.
- The API was exposed via LangServe, with CORS support.

Challenges and Solutions

One of the main challenges was finding the right balance in information retrieval parameters. Initially, k=2 was insufficient, leading to an increase to k=5. However, after further testing, k=3 was found to provide more accurate responses while avoiding redundant information.

Another major issue was system latency. Since the chatbot does not implement streaming for responses, the user may perceive a longer delay, as the text generation happens entirely before being sent.

Using LangServe also posed additional complications, as the tool is now deprecated and no longer receives updates. Integrating it into the project required adapting solutions to outdated documentation and solving problems caused by the lack of official support. While it allowed for a quick model deployment, it might be worth considering more modern, community-supported alternatives in the future such as LangGraph.

Another critical aspect was managing credentials and sensitive data. To prepare the code for deployment in a public repository and its eventual implementation in AWS, best practices were adopted for secure API key storage. Measures such as environment variables and secret files mounted in containers were implemented to prevent credential exposure in the source code. This approach minimized security risks and ensured that the keys remained protected throughout the development and deployment process.

Potential Future Improvements and Conclusion

- Dynamic adjustment of parameters (*k* and language model) to optimize accuracy and efficiency.
- Conversation history to enhance coherence in consecutive responses.
- Deployment of a custom frontend, instead of relying on LangServe's Playground.

The final solution provides a functional and effective chatbot for retrieving information about Promptior, with a modular design that allows for future improvements and expansions.

I must highlight that this technical test was highly enriching, as it motivated me to develop and deploy a project from scratch, something I hadn't done in a long time. It allowed me to experiment and put my skills to the test, particularly in problem-solving, consulting reliable sources, and formulating effective solutions based on the knowledge I have gained throughout my professional career.

Diagrama de componentes - Component Diagram