Trabajo práctico final

FCEIA

Procesamiento del Lenguaje Natural

Enero, 2024.

Integrantes:

Garay Gorta, Isaías.

Docentes:

Cavallo, Andrea.

D'Alessandro, Ariel.

Geary, Alan.

Manson, Juan Pablo.

Ejercicio 1

Fuente de datos

Para la implementación del **RAG** se utilizaron como fuentes de datos los siguientes recursos:

- Archivos `pdf` de la documentación oficial de Python.
- Archivo `csv` sobre repositorios populares de Python.
- Base de datos de grafos online DBpedia.

A los archivos `pdf` se les extrajo el texto y se normalizo para luego vectorizarlos y almacenarlos en una base de datos ChromaDB. Respecto a los datos tabulares del `csv` se decidió quedarse solamente con las entradas del año 2023 debido a que el archivo es muy extenso. Por último se usará la base de datos externa para aquellas búsquedas relacionadas con programación pero no específicamente sobre Python.

Documentos de texto

A los strings que se obtienen de los archivos pdf se les aplica una limpieza en las que se remueven los acentos y las *stop words* del texto, luego se divide este texto en fragmentos para mejorar los resultados del modelo, finalmente se realiza un *embedding* para obtener el significado semántico del texto y se almacenan estos embeddings en una base de datos *ChromaDB*.

Base de datos de grafos

Para esta parte se utilizará la base de grafos online DBpedia, esta servirá para brindarle contexto al modelo cuando el usuario haga preguntas sobre conceptos más generales sobre programación o sobre otros lenguajes que no sean Python, en los ejemplos del código vemos que es posible preguntarle sobre el lenguaje *Java* gracias a que puede obtener la información de contexto pertinente desde la base de datos de grafos.

Datos tabulares

De los datos tabulares se obtendrá la descripción del proyecto si es posible obtener el nombre del proyecto desde el prompt del usuario en otro caso se usara todo el dataframe como un string para brindarle contexto al modelo.

Elección de la fuente de datos.

Se utilizará el modelo pre-entrenado `paraphrase-MiniLM-L6-v2` para hacer el embedding de la consulta del usuario y calcular la similaridad con las consultas de ejemplo, de esta forma se puede elegir cual de las tres fuentes externas es más probable que responda la pregunta original.

RAG

La 'generación aumentada por recuperación' (RAG) consiste en proporcionar a los modelos de lenguaje de gran escala (LMM) información de contexto adicional sobre algún tema en particular. Esto permite que los modelos puedan producir respuestas más exactas.

Finalmente se puede apreciar que el sistema responde de forma satisfactoria a las consultas usando solamente la información que obtiene del prompt de contexto que se le brinda.

Conclusión

En este proyecto, se ha demostrado la eficacia de la Generación Aumentada por Recuperación (RAG) como método para mejorar la capacidad de los modelos de lenguaje de gran escala (LMM) para generar respuestas precisas y contextualmente relevantes. A través de la integración de diversas fuentes de datos, como documentos de texto, bases de datos de grafos y datos tabulares, se ha enriquecido el conocimiento del modelo, permitiéndole responder a una amplia gama de consultas relacionadas con la programación, específicamente en el contexto de Python y otros lenguajes.

La extracción, limpieza y vectorización de datos de documentos `pdf` y archivos `csv`, junto con el uso de bases de datos externas como DBpedia, ha proporcionado al modelo un sólido conjunto de información contextual. Este enfoque ha permitido al modelo comprender y responder de manera efectiva a preguntas sobre conceptos de programación.

Ejercicio 2

Los sistemas de agentes inteligentes son estructuras diseñadas para interactuar con su entorno, procesar información y tomar decisiones en función de los datos disponibles. Entre estos, los modelos de lenguaje de gran escala (LLM) destacan como exponentes de la inteligencia artificial capaces de manejar y generar lenguaje natural en gran volumen. Estos modelos son entrenados con extensas cantidades de datos textuales, lo que les permite comprender y producir contenido lingüístico de manera efectiva y precisa.

Por ende, un agente que se fundamenta en modelos de lenguaje de gran escala (LLM) muestra un potencial significativo para abordar una amplia gama de problemas prácticos cuando se le coloca en un entorno pertinente. En la actualidad, se encuentran disponibles diversos modelos, tanto de carácter privado como de código abierto, lo que amplía las opciones para su aplicación en distintos ámbitos y contextos.

Algunos ejemplos de agentes inteligentes de propiedad privada incluyen ChatGPT y Bard. Estos modelos son altamente valorados y ampliamente adoptados a nivel mundial. Poseen la capacidad de abordar tanto situaciones simples como complejas planteadas por los usuarios en diversos contextos, utilizando exclusivamente lenguaje natural humano, ya sea hablado o escrito.

Últimamente, los modelos de agentes inteligentes han evolucionado mediante la integración con otros sistemas, permitiendo la orquestación de múltiples agentes para resolver problemas complejos. Estas integraciones incluyen capacidades como la carga y análisis de archivos, que posibilitan a los usuarios subir documentos o imágenes y obtener resúmenes, categorías o etiquetas relevantes. Además, se han incorporado herramientas externas como navegación web, análisis de datos y sistemas de generación y reconocimiento de imágenes. Sin embargo, estos sistemas multiagente actualmente están disponibles únicamente para usuarios de pago, careciendo de acceso libre o código abierto para el público en general.

A pesar de esto, hay proyectos de código abierto en desarrollo que buscan crear sistemas multiagentes similares. Un ejemplo es AutoGPT, un proyecto en Github que combina el modelo de lenguaje similar a ChatGPT con técnicas de auto-prompting para mejorar la autonomía del modelo en la resolución de problemas genéricos.

Algunos de los casos de uso para este tipo de sistemas son los siguientes:

Salud

Los LLMs multiagentes pueden proporcionar expertise bajo demanda en áreas como diagnósticos y opciones de tratamiento, mejorando la atención al paciente y los resultados médicos

Finanzas

Las instituciones financieras pueden emplear LLMs multiagentes para analizar tendencias del mercado, evaluar estrategias de inversión y ofrecer asesoramiento financiero personalizado, mejorando sus servicios y la satisfacción del cliente.

Educación

Estos sistemas pueden revolucionar la educación al brindar a los estudiantes acceso a expertos en diversas materias, ofreciendo experiencias de aprendizaje personalizadas y fomentando el crecimiento académico.

Planteo de problemática

La problemática planteada es la de permitir que personas que estén dando sus primeros pasos en programación puedan usar un sistema multiagente para obtener respuestas a sus dudas, sugerencias y *feedback* para mejorar el código que escriben.

Para lograr esto considero que sería necesario contar con dos agente, uno de ellos sería el encargado de generar el código Python para responder las consultas del usuario, deberá tener acceso a un intérprete del lenguaje para poder ejecutar el código del usuario y los mismo mensajes de error del propio intérprete para ayudar a resolver el problema.

El segundo agente será intermediario entre el usuario y el agente especializado en analizar y generar código, este segundo agente prepara el prompt con contexto adicional en caso de ser necesario, deberá tener acceso a internet para buscar información sobre Python o alguna de sus librerías.

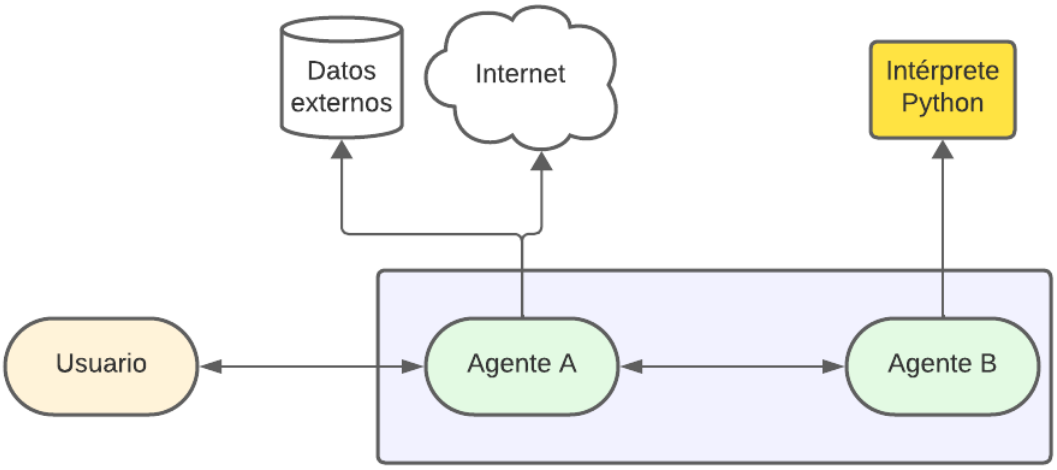

Figura 1: Esquema del sistema multiagente planteado

Fuentes

- Intelligent agent - Wikipedia
- Application of Pretrained Large Language Models in Embodied Artificial Intelligence
- Revolutionizing AI: The Era of Multi-Agent Large Language Models