TRABAJO PRÁCTICO FINAL

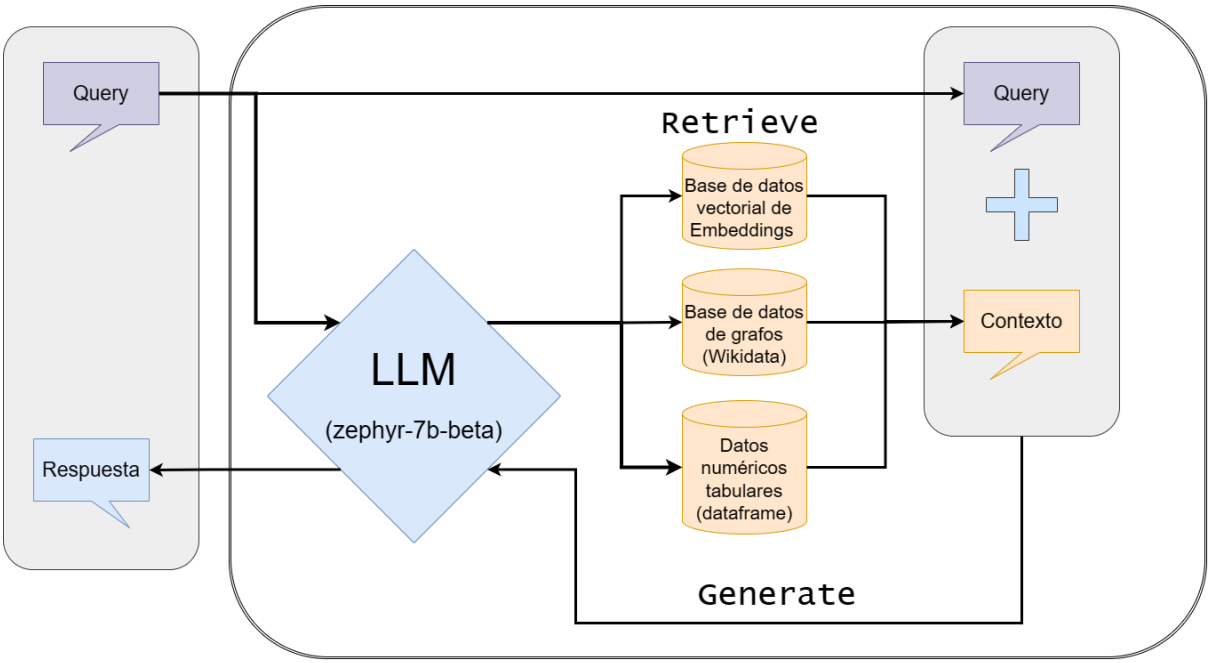
PROCESAMIENTO DEL LENGUAJE NATURAL

Tecnicatura Universitaria en Inteligencia Artificial
FECIA - UNR
18/12/23

Leandro Salvañá

1. Ejercicio 1: Implementación de RAG

El objetivo de este RAG (Retrieval Augmented Generation) es crear un chatbot experto en **Harry Potter**. El esquema general es el siguiente:



Se utilizará un LLM con un prompt con formato Few-Shot para clasificar qué fuente de datos utilizar dado el prompt introducido, luego se usan diversos métodos para traer el contexto adecuado de cada una de estas fuentes y finalmente se alimenta el LLM con el prompt del usuario sumado al contexto y las instrucciones de que no utilice su conocimiento previo sino el contexto brindado. El enunciado indica que el intercambio debe ser en idioma Español. Debido a que los dataset están en inglés y que los modelos en general funcionan mejor en este idioma, se decidió traducir la query al inglés, trabajar en este idioma y luego traducir al español la respuesta.

1.1. Fuentes de datos

Para comenzar la implementación del RAG se prepararon las fuentes de datos para brindar contexto a los prompts. Estas serán una base de datos vectorial ChromaDB llena con embeddings generados a partir de los siete libros de la saga principal de Harry potter, Wikidata que es una base de datos de grafos y datos tabulares ubicados en formato CSV.

1.1.1. Texto

Los libros de Harry Potter fueron descargados de un dataset de Kaggle y el texto se extrajo con la librería PyPDF2.

Se limpiaron los saltos de línea "ficticios" es decir, que existen sólo porque se llegó al extremo de la hoja del PDF y no porque tenga un sentido semántico. Esto es porque el splitter de texto de langchain utiliza los saltos de línea como guía para dividir el texto.

1.1.2. Datos numéricos tabulares

Los datos tabulares se obtuvieron de un dataset de Kaggle y consta de un CSV con información sobre las novelas físicas. Contienen información del nombre del libro, copias vendidas en el reino unido, copias vendidas en todo el mundo, fecha de publicación, mes de publicación, año de publicación, cantidad de páginas, cantidad de palabras, duración del audiolibro y cantidad de premios ganados. Los datos fueron cargados en un dataframe de Pandas y luego se hizo una función para imprimirlo entero en formato de texto con sentido semántico.

Quizás hubiera sido mejor dar como contexto sólo la información solicitada en el prompt. Pero por priorizar el tiempo de realización y dado el reducido tamaño del dataset, se optó por brindar todo el texto cuando el prompt hace referencia a las novelas físicas.

1.1.3. Base de datos de grafos

Se utilizó una base de datos de grafos online, siendo Wikidata la opción elegida. Se decidió que se brindará contexto de esta fuente únicamente cuando se pregunte por un personaje.

Para esto, se definió una función que pide a la base de datos el ID del sujeto del personaje al que el prompt hace referencia, y luego se le piden todos las etiquetas de objetos relativos al sujeto dada una lista de IDs de predicados previamente investigados.

```python
# ID de Albus Dumbledore en Wikidata
sujeto_label_ejemplo = "Albus Dumbledore"

# Obtener y mostrar los valores asociados a la propiedad
propiedades_personaje = obtener_valores_por_sujeto(sujeto_label_ejemplo)

print(propiedades_personaje)
```

```
Albus Dumbledore  :
sex or gender: male
country of citizenship: United Kingdom
name in native language: Albus Percival Wulfric Brian Dumbledore
birth name: Albus Percival Wulfric Brian Dumbledore
given name: Percival
given name: Brian
given name: Albus
given name: Wulfric
date of birth: 1881-01-01T00:00:00Z
place of birth: Mould-on-the-Wold
date of death: 1997-06-30T00:00:00Z
manner of death: homicide
cause of death: Avada Kedavra
killed by: Severus Snape
father: Percival Dumbledore
mother: Kendra Dumbledore
sibling: Aberforth Dumbledore
sibling: Ariana Dumbledore
```

1.1.4. Base de datos vectorial de embeddings

1.1.4.1. Dividir los textos

Una vez extraídos los textos de los libros desde los PDFs, se procedió a dividir los textos en fragmentos “chunks” con una longitud significativa para captar el contexto. Al tratarse de libros de narración de fantasía con mucha adjetivación se tomó un fragmento de longitud considerable: 1000 caracteres.

1.1.4.2. Limpieza de fragmentos

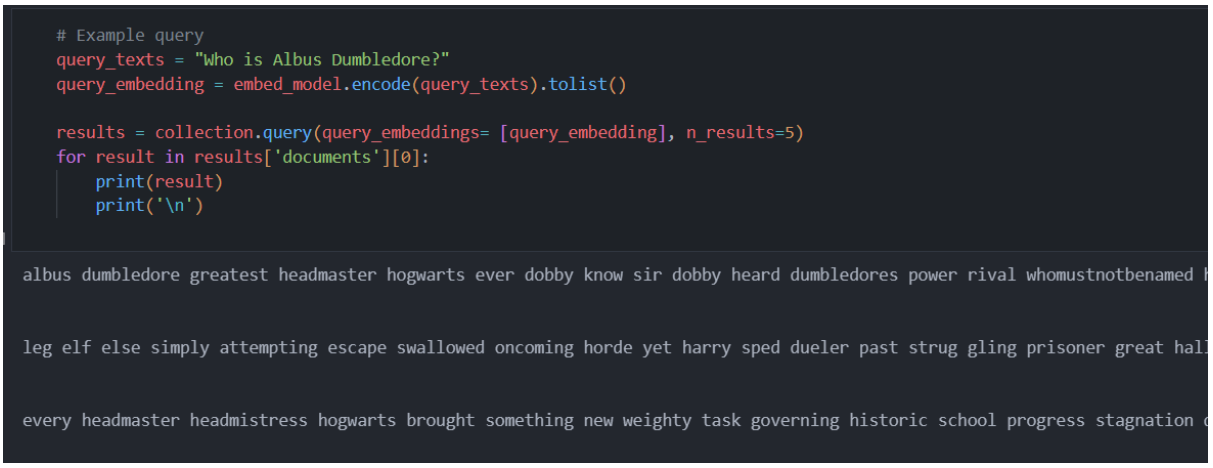
Para mejorar el desempeño del modelo de embeddings, se limpiaron los fragmentos del texto. Esta limpieza consistió en pasar todo el texto a minúsculas, la eliminación de símbolos y stopwords y lematizar.

1.1.4.3. Embeddings

Una vez limpios los fragmentos se utilizó el modelo de embeddings paraphrase-multilingual-mpnet-base-v2 para realizar la vectorización de los fragmentos.

1.1.4.4. Almacenar embeddings en una base de datos ChromaDB

Ahora, obtenidos los embeddings, se guardaron en una base de datos ChromaDB junto con los fragmentos sin limpiar. Para luego obtener los embeddings de un prompt y traer los vectores más cercanos como se observa en la siguiente imagen.



```
# Example query
query_texts = "Who is Albus Dumbledore?"
query_embedding = embed_model.encode(query_texts).tolist()

results = collection.query(query_embeddings= [query_embedding], n_results=5)
for result in results['documents'][0]:
    print(result)
    print('\n')
```

```
albus dumbledore greatest headmaster hogwarts ever dobby know sir dobby heard dumbledores power rival whomustnotbenamed

leg elf else simply attempting escape swallowed oncoming horde yet harry sped dueler past strug gling prisoner great hal

every headmaster headmistress hogwarts brought something new weighty task governing historic school progress stagnation
```

1.2. Implementación de RAG

1.2.1. Traducción de prompts y respuestas

Como se adelantó en la introducción, el enunciado pide que el intercambio sea en español, pero dado a que los dataset están en inglés y el mejor desempeño de los modelos en este idioma, se optó por traducir el prompt al inglés, trabajar en la lengua sajona y luego traducir la respuesta al español nuevamente.

Para llevar a cabo esto se utilizaron los modelos "opus-mt-es-en" y "opus-mt-es-en" de Helsinki-NLP disponibles en HuggingFace.

1.2.2. Clasificación de prompts para elegir la fuente de contexto

Para clasificar las preguntas y elegir la fuente para dar contexto, se utilizó un LLM específicamente "zephyr-7b-beta" diponible en Hugging face y un formato de prompt Few-Shot para obtener respuestas más exactas. De esta manera al introducir un prompt relacionado a un personaje, no sólo indicará que se trata de una pregunta sobre un personaje sino que además devuelve el nombre del personaje como se muestra en el ejemplo a continuación.

```
# Ejemplo)
prompt = 'Where does Minerva MacGonagall work?'
answer = choose_data_source(prompt, api_key=api_key)
print('\nRESPUESTA: \n', answer)
```

```
RESPUESTA:
 "Personaje: [Minerva McGonagall]" for questions or tasks about Minerva McGonagall's identity, such as where she works.
```

1.2.3. Interpretar la respuesta

Ya que se le pidió un formato de respuesta muy específico al LLM, se puede utilizar una función muy sencilla con expresiones regulares para obtener la información necesaria. La función formatea la información en una lista con dos elementos. El primero con el tema de la pregunta ("Personaje", "Novelas", "Historia") y el segundo en `None` a menos que sea una pregunta sobre un personaje, dónde también se obtendrá el personaje en cuestión y se colocará en esta posición para realizar la query en la base de datos de grafos. A continuación, un ejemplo:

```
# Ejemplo de uso:
respuesta_limpia = answer_cleaner(answer)
print(respuesta_limpia)
```

```
4]
['Personaje', 'Minerva McGonagall']
```

1.2.4. Devolver el contexto

Teniendo identificado el tipo de prompt y, por ende, la fuente que debe dar el contexto, se armó una función que se le pasa el binomio resultante de la función anterior y devuelve el texto predefinido del dataframe de los libros, vectoriza y busca el prompt en la ChromaDB o hace el proceso descrito anteriormente para obtener los datos de un personaje en la base de datos de grafos. Continuando con el ejemplo con el que se venía trabajando:

```python
    # Ejemplo
    contexto = devolver_contexto(respuesta_limpia, prompt)[0]

    print(contexto)
```

```
Minerva McGonagall  :
sex or gender: female
country of citizenship: United Kingdom
name in native language: Minerva McGonagall
given name: Minerva
date of birth: 1935-10-04T00:00:00Z
occupation: professor
occupation: employee
occupation: head teacher
occupation: school teacher
employer: Hogwarts
employer: Department of Magical Law Enforcement
position held: Headmaster of Hogwarts
member of: Order of the Phoenix
member of: Gryffindor
```

1.2.5. Preguntar al agente con el contexto agregado

El último paso es darle al mismo LLM que ayudó a clasificar el prompt, el prompt original y el contexto, y pedirle que no use conocimiento previo.

1.2.6. Resultados

A continuación se muestra un ejemplo para una situación dónde se debe utilizar cada una de las fuentes de datos.

```
--------------------------------------------------------------------
PREGUNTA:
Qué pasó en la cámara de los secretos?
--------------------------------------------------------------------
FUENTE ESCOGIDA: Base de datos vectorial de Embeddings
--------------------------------------------------------------------
CONTEXTO BRINDADO:
nooooooo scene whirled darkness became complete harry felt falling crash landed spreadeagl
--------------------------------------------------------------------
RESPUESTA:
En la cámara de los secretos, Harry encontó el diario de Riddle, se desmayó y se desplomó
```

```
--------------------------------------------------------------------------
PREGUNTA:
¿Cuáles son los hermanos de Ron Weasley?
--------------------------------------------------------------------------
FUENTE ESCOGIDA: Base de datos de grafos
--------------------------------------------------------------------------
CONTEXTO BRINDADO:
Ron Weasley  :
sex or gender: male
country of citizenship: United Kingdom
name in native language: Ron Weasley
given name: Ron
given name: Ronald
date of birth: 1980-03-01T00:00:00Z
place of birth: Devon
father: Arthur Weasley
mother: Molly Weasley
sibling: Ginny Weasley
sibling: Bill Weasley
sibling: Percy Weasley
sibling: Charlie Weasley
sibling: Fred Weasley
sibling: George Weasley
occupation: student
occupation: entrepreneur
...

--------------------------------------------------------------------------
RESPUESTA:
Los hermanos de Ron Weasley hijo Bill, Percy, Charlie, Fred y Ginny Weasley.
```

```
PREGUNTA:
Cuántas unidades se vendieron de "La cámara secreta"?
--------------------------------------------------------------------------
FUENTE ESCOGIDA: Dataframe
--------------------------------------------------------------------------
CONTEXTO BRINDADO:
Book Name: Harry Potter and the Sorcerer's Stone
Copies Sold in UK: 4.2 millons
Copies Sold WorldWide: 120 millons
Publish Date: 26 June 1997
Pages: 223
Words: 76,944
Audiobook: 9 hrs and 33 mins
Total Awards Won: 8

Book Name: Harry Potter and the Chamber of Secrets
Copies Sold in UK: 3.5 millons
Copies Sold WorldWide: 77 millons
Publish Date: 2 July 1998
Pages: 251
Words: 85,141
Audiobook: 11 hrs and 5 mins
Total Awards Won: 5
...

--------------------------------------------------------------------------
RESPUESTA:
Se vendieron 3.5 millones de unidades de "La cámara secreta" en el Reino Unido, y se vendieron 77 millones de unidades
```

2. Ejercicio 2: Sistema Multiagente

2.1. Modelos de lenguaje multiagente

El LLM (Large Language Model) multiagente, en esencia, implica la fusión de múltiples agentes impulsados por LLMs que trabajan en conjunto. A diferencia de la entidad singular tradicional que proporciona respuestas, los sistemas multiagente constan de varios agentes de IA, cada uno especializado en distintos dominios, lo que contribuye a la resolución integral de problemas. Esta sinergia colaborativa da como resultado soluciones más matizadas y efectivas.

2.2. Ventajas

- **Experiencia mejorada:** Cada agente dentro del sistema posee conocimientos especializados en su campo, lo que garantiza respuestas profundas y precisas.

- **Mejora de la resolución de problemas**: Los LLM multiagente destacan al aprovechar la inteligencia colectiva de sus agentes, abordando desafíos complejos desde múltiples perspectivas.

- **Robustez y fiabilidad**: Estos sistemas mitigan el riesgo de errores al contar con redundancia y fiabilidad. Si un agente encuentra un problema, otros pueden intervenir, asegurando funcionalidad continua y reduciendo la probabilidad de errores.

- **Adaptabilidad**: Los LLM multiagente pueden evolucionar con el tiempo, integrando nuevos agentes para abordar desafíos emergentes. Esta flexibilidad los hace adecuados para diversas aplicaciones, desde la atención médica hasta las finanzas, en un mundo dinámico.

2.3. Proyectos de sistemas multiagente

2.3.1. AutoGPT

AutoGPT redefine la interacción entre la inteligencia artificial y el usuario, eliminando la necesidad de intercambios continuos. A diferencia de ChatGPT, que requiere solicitudes sucesivas, AutoGPT solo precisa un mensaje inicial del usuario. A partir de este, el agente genera una lista de tareas para cumplir con la solicitud sin necesidad de más información. El sistema, complejo y basado en múltiples componentes, se conecta a internet, utiliza gestión de memoria a corto y largo plazo (Long Short-Term Memory), GPT-4 para generación de texto avanzada, y GPT-3.5 para almacenamiento y resumen de archivos. Para

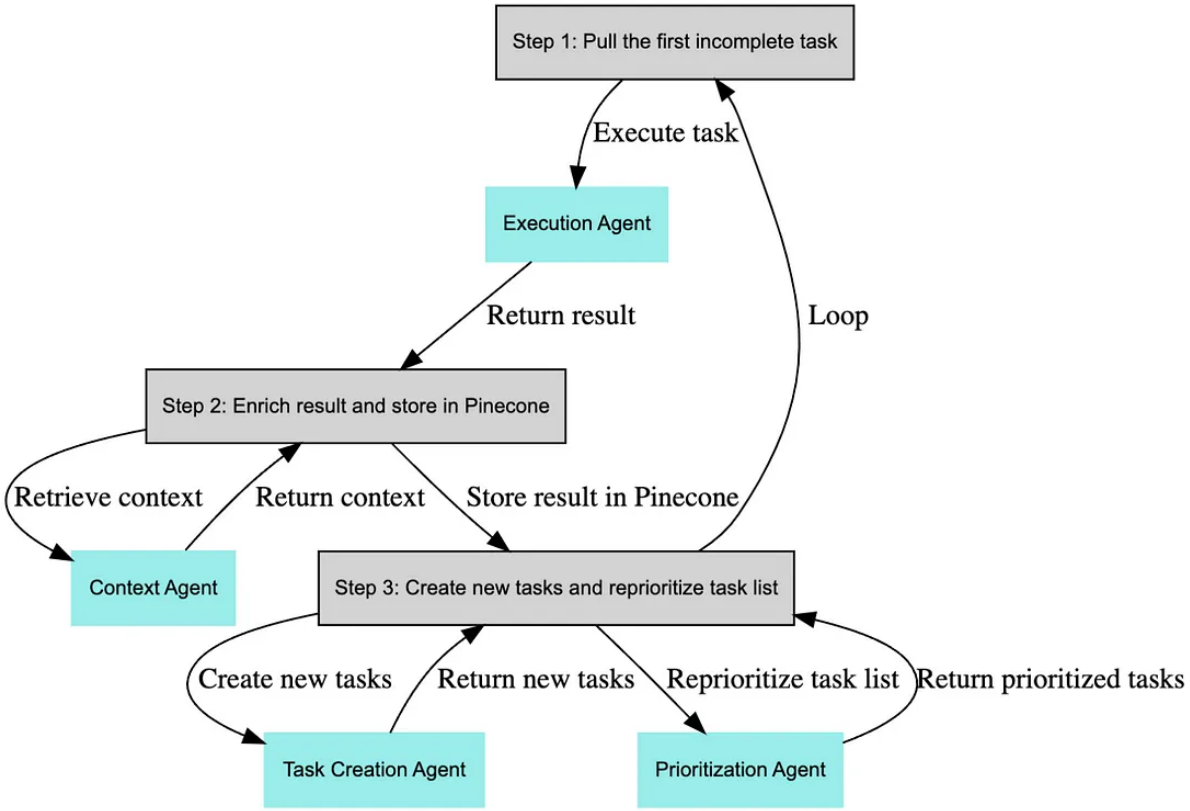
ejecutar AutoGPT, el usuario proporciona nombre, rol y objetivos, y el agente trabaja para cumplir con dichas metas.

2.3.2. BabyAGI

BabyAGI, al igual que AutoGPT, se enfoca en desarrollar agentes autónomos con habilidades de planificación a largo plazo y uso de memoria mediante técnicas de aprendizaje por refuerzo. Esta versión simplificada demuestra cómo la inteligencia artificial puede mejorar la eficiencia en la gestión de tareas.

En cuanto a su funcionamiento, el script de BabyAGI opera en un bucle infinito que sigue estos pasos:

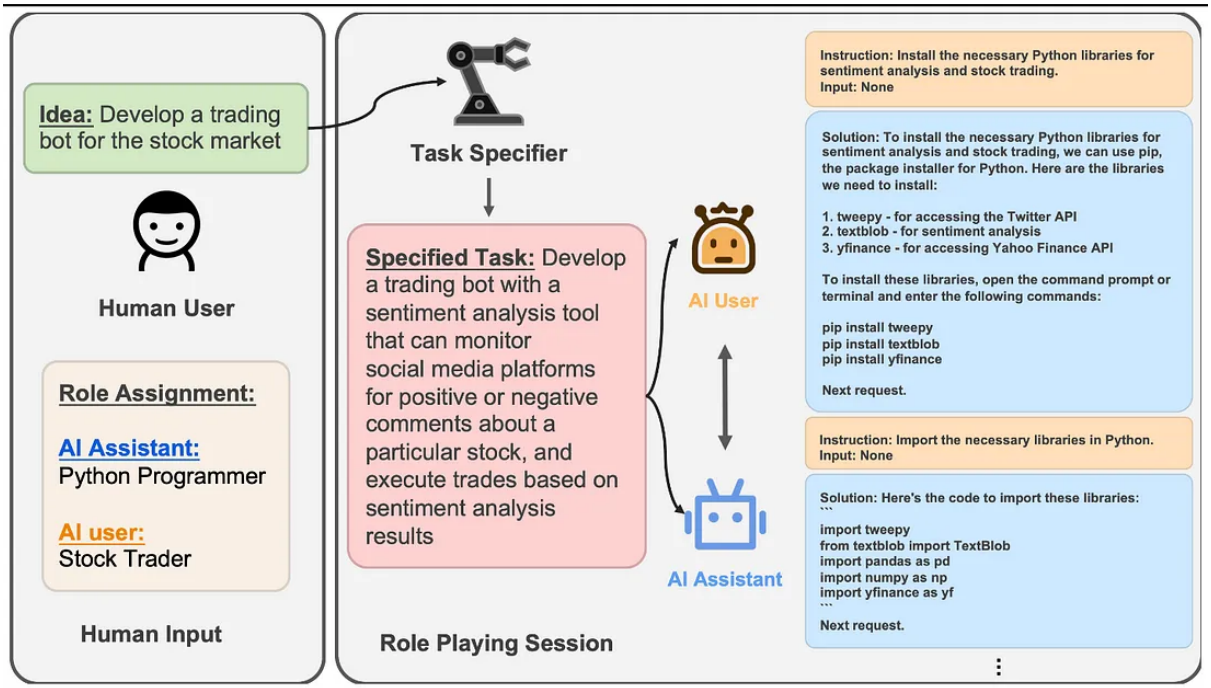
1. Extrae la primera tarea de la lista de tareas.
2. Envía la tarea al agente de ejecución, que utiliza la API de OpenAI para completarla según el contexto.
3. Enriquece el resultado y lo almacena en Pinecone.
4. Crea nuevas tareas y reorganiza la lista de tareas según el objetivo y el resultado de la tarea anterior.



BabyAGI integra las capacidades de procesamiento del lenguaje natural (NLP) de OpenAI para crear nuevas tareas basadas en el objetivo predefinido. Pinecone se utiliza para almacenar y recuperar los resultados, proporcionando contexto para futuras tareas.

2.3.3. CAMEL

CAMEL es un proyecto que busca la colaboración entre dos agentes con personalidades distintas en un entorno de simulación específico. Estos agentes cuentan con memoria a corto y largo plazo, utilizando pasos de reflexión para asignar puntuaciones de importancia a las observaciones y generalizar lo aprendido. La novedad de CAMEL reside en la interacción colaborativa entre dos agentes dentro de un entorno simulado.



2.3.4. Generative Agents

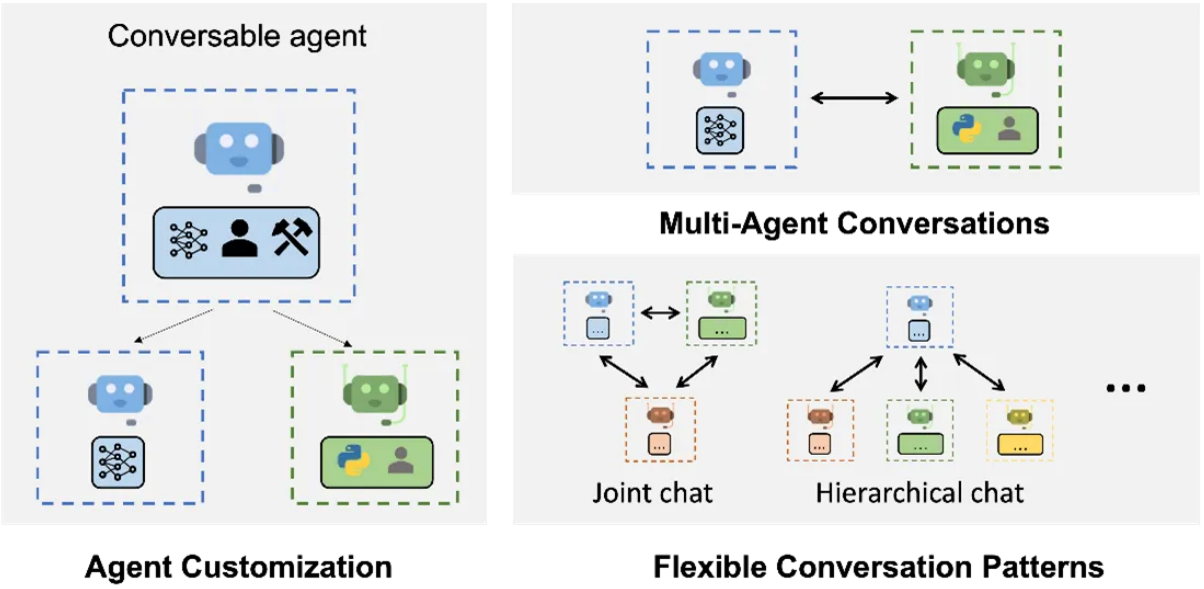
Generative Agents es un proyecto enfocado en la creación de entornos de simulación para agentes autónomos. Estos agentes se basan en modelos generativos para imitar comportamientos individuales y grupales similares a los humanos, fundamentándose en sus "identidades, entorno y experiencias cambiantes".

En un estudio, investigadores de Stanford y Google desarrollaron un entorno interactivo inspirado en el popular videojuego de simulación The Sims. Este entorno tipo "sandbox" permitió analizar los comportamientos individuales y sociales de los agentes de inteligencia artificial, revelando que algunos agentes, como la Agente Alice, mostraron preferencias y objetivos similares a los humanos, como la pintura.

Generative Agents sigue una arquitectura con componentes como observación, reflexión y planificación. Pueden almacenar registros de las experiencias de un agente mediante lenguaje natural, y los usuarios pueden interactuar con ellos a través del mismo. Estos agentes también tienen la capacidad de realizar diversas tareas, como preparar desayunos, ir al trabajo, pintar, formar relaciones y entablar conversaciones, mostrando similitudes sorprendentes con los humanos en términos de recordar, recuperar y reflexionar sobre el pasado y el presente.

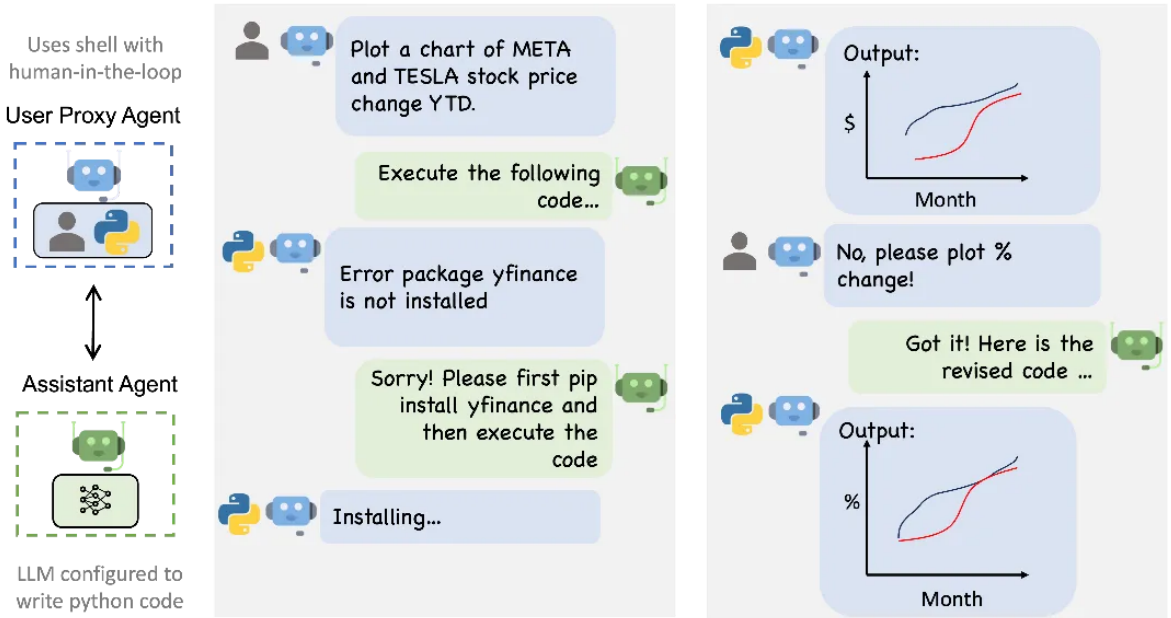
2.3.4. AutoGen

AutoGen es un proyecto de Microsoft que permite a los usuarios crear múltiples agentes autónomos similares a ChatGPT para colaborar en tareas específicas. Este marco flexible permite definir varios agentes, asignarles roles y orquestar sus esfuerzos de colaboración. Permite flujos de trabajo complejos basados en Modelos de Lenguaje de Aprendizaje (LLM), lo que simplifica la orquestación, optimización y automatización de estos flujos. AutoGen Facilita la integración de varios agentes en proyectos sin cambiar el código base existente y puede sustituir la API de OpenAI.



Características principales de AutoGen:

1. **Marco para flujos de trabajo de LLM**: Simplifica la orquestación de flujos de trabajo basados en grandes modelos de lenguaje.
2. **Agentes personalizables:** Ofrece agentes personalizables que utilizan avanzados LLM como GPT-4, integrándose con humanos y herramientas para abordar limitaciones.
3. **Chat automatizado:** Los agentes pueden realizar conversaciones automatizadas, mejorando la eficiencia en la colaboración.
4. **Integración con otros LLM**: Funciona con la API de OpenAI y puede integrarse con cualquier modelo de lenguaje grande mediante API.



El sistema AutoGen emplea dos agentes de IA para mejorar la interacción del usuario. El agente proxy de usuario actúa como la encarnación digital del usuario, automatizando decisiones y tareas, con flexibilidad para involucrar al usuario humano cuando sea necesario. El Agente Asistente es la fuente principal de inteligencia y resolución de problemas, generando salidas y proporcionando soluciones. La colaboración entre estos dos agentes optimiza la eficiencia mediante la automatización y conserva la capacidad de supervisión humana, equilibrando la automatización con el toque humano. El diseño permite tener múltiples agentes y agentes proxy de usuario para modelar diversos procesos de negocio.

2.4. Conclusión

El impacto de los Modelos de Lenguaje Multiagente en la inteligencia artificial es evidente en el surgimiento de proyectos como AutoGPT, BabyAGI, CAMEL, Generative Agents y AutoGen, que están transformando la forma en que los agentes autónomos aprenden y colaboran en entornos diversos. Estos avances están marcando una nueva era en la capacidad de los agentes autónomos para la planificación a largo plazo y la utilización eficiente de la memoria.

A la vanguardia de esta revolución se encuentra AutoGen, el proyecto de Microsoft que redefine las posibilidades de la inteligencia artificial. Permitiendo la creación de sistemas multiagente altamente colaborativos, AutoGen no solo mejora la productividad, sino que también desencadena una ola de creatividad en campos como el desarrollo de software, la investigación, la escritura creativa y el análisis de datos.

En resumen, el afloramiento de proyectos como AutoGPT, BabyAGI, CAMEL, Generative Agents y AutoGen señala que el futuro de la inteligencia artificial ya está aquí. Estamos presenciando un panorama donde los agentes autónomos son autónomos, adaptables e increíblemente inteligentes, allanando el camino para soluciones innovadoras en una variedad de campos complejos.

2.5. Problemática

2.5.1. Problemática a Solucionar con el Sistema Multiagente

La problemática que se va a abordar con el sistema multiagente es automatizar la interacción entre un agente asistente (capaz de generar código) y un agente proxy de usuario (encargado de ejecutar el código y proporcionar retroalimentación como GPT-4). La tarea específica podría ser cualquier fin que se pueda lograr mediante la programación en Python. Podría ser hacer un análisis de datos a un dataset o consultar y comparar los cambios en los precios

de las acciones YTD (Year-To-Date) de las empresas META y TESLA mediante la consulta de librerías financieras de Python.

2.5.2. Definición de Agentes Involucrados

Agente Asistente

- Es un agente basado en un modelo de lenguaje (LLM) capaz de generar código Python en respuesta a las solicitudes del usuario.
- Su función principal es proponer soluciones codificadas para la tarea encomendada.
- Puede analizar y depurar código, así como generar nuevo código en caso de errores.

Agente Proxy de Usuario

- Actúa como intermediario entre el usuario y el agente asistente.
- Ejecuta el código proporcionado por el agente asistente y devuelve la salida o resultados al mismo.
- Determina la terminación de la conversación.

2.5.3. Esquema del Sistema Multiagente

El sistema multiagente sigue un flujo de interacción donde el agente asistente y el agente proxy de usuario se comunican para resolver la tarea encomendada. Aquí está el esquema general del sistema:

1. **Inicio de Conversación**
   - El agente proxy de usuario inicia la conversación con el agente asistente, proporcionando una tarea específica.

2. **Generación y Ejecución de Código**
   - El agente asistente propone soluciones codificadas para la tarea encomendada, generando código Python.
   - El agente proxy de usuario ejecuta el código proporcionado y recopila la salida.

3. **Retroalimentación y Depuración**
   - Si la ejecución del código es exitosa, el agente proxy de usuario devuelve la salida al agente asistente.
   - Si hay errores en la ejecución, el agente proxy de usuario proporciona información de error al agente asistente, que a su vez puede generar nuevo código corregido.

4. **Iteración**

- La comunicación entre el agente asistente y el agente proxy de usuario continúa iterativamente hasta que se completa la tarea o se alcanza un límite predefinido de respuestas automáticas.

2.6. Fuentes

1. Microsoft - AutoGen: Habilitación de aplicaciones de modelos de lenguaje de gran tamaño de próxima generación
    - Microsoft. AutoGen: Enabling Next-Generation Large Language Model Applications. Recuperado de https://www.microsoft.com/en-us/research/blog/autogen-enabling-next-generation-large-language-model-applications/

2. Medium - Revolucionando la IA: la era de los grandes modelos de lenguaje multiagente
    - Fowler, G. A. Revolutionizing AI: The Era of Multi-Agent Large Language Models. Medium. Recuperado de https://gafowler.medium.com/revolutionizing-ai-the-era-of-multi-agent-large-language-models-f70d497f3472

3. Medium - Agentes Autónomos y Simulaciones en LLM: Un vistazo a AutoGPT, BabyAGI, CAMEL y Generative Agents
    - Avila, D. (Fecha de publicación). Agentes Autónomos y Simulaciones en LLM: Un vistazo a AutoGPT, BabyAGI, CAMEL y Generative Agents. Medium. Recuperado de https://medium.com/latinxinai/agentes-aut%C3%B3nomos-y-simulaciones-en-llm-un-vistazo-a-autogpt-babyagi-camel-y-generative-agents-bdb0bbfcddac

4. Medium - AutoGen paso a paso de Microsoft: el futuro de la programación con agentes autónomos similares a GPT
    - Lee, E. (Fecha de publicación). AutoGen Step-by-Step by Microsoft: The Future of Programming with Autonomous GPT-Like Agents. Medium. Recuperado de https://drlee.io/step-by-step-autogen-by-microsoft-the-future-of-programming-using-autonomous-gpt-like-agents-105ac45a518f