

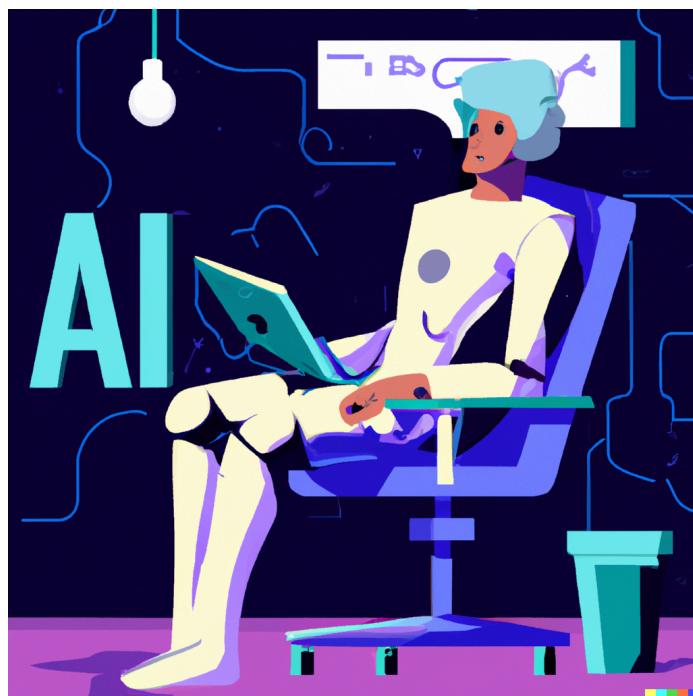


Procesamiento del Lenguaje Natural

Tema: "Conversaciones Épicas: Explorando la Sabiduría de la Tierra Media a través de un Chatbot Tolkieniano".

Docentes: Juan Pablo Manson, Alan Geary y Andrea Carolina Leon Cavallo.

Alumno: Alejo Lo Menzo.



Índice:

Ejercicio 1: Implementación del Chatbot.....	2
Introducción:.....	2
Descarga e importación de librerías:.....	2
Carga del token de Hugging Face:.....	2
Extracción de archivos:.....	3
Extracción de texto:.....	3
Datos tabulares:.....	4
Base de datos de grafos:.....	5
Base de datos vectorial de embeddings:.....	5
Implementación del RAG:.....	6
Traducción de prompts y respuestas:.....	6
Clasificación para elegir la fuente de contexto:.....	7
Interpretación de respuesta:.....	7
Devolución de contexto:.....	8
Preguntar al agente con el contexto aclarado:.....	8
Uso de la técnica RAG:.....	9
Resultado final:.....	10
Conclusión:.....	10
Ejercicio 2: Agentes inteligentes.....	11
¿Qué es un agente inteligente impulsado por LLM?.....	11
¿Por qué se destacan los agentes de LLM?.....	11
Casos de uso para agentes de LLM:.....	12
Único agente:.....	12
Sistema multiagente:.....	13
Cooperación de persona - agente:.....	13
Investigación sobre las aplicaciones que actualmente ofrecen agentes de IA para distintos objetivos:.....	14
Problemática a solucionar con un sistema multiagente:.....	15
Interacción entre los agentes:.....	15
Esquema del Sistema Multiagente:.....	16
Informe de la investigación:.....	17
Bibliografía:.....	18

Ejercicio 1: Implementación del Chatbot

Introducción:

El proyecto propuesto tiene como objetivo la creación de un chatbot experto en el fascinante mundo de "El Señor de los Anillos" y "El Hobbit" utilizando la técnica Retrieval Augmented Generation (RAG). Este sistema permitirá a los usuarios entablar conversaciones en español con el chatbot, formulando preguntas sobre diversos aspectos del universo literario creado por J.R.R. Tolkien.

Descarga e importación de librerías:

Para la utilización del Chatbot debemos instalar diversas bibliotecas y herramientas mediante el comando "`!pip install`". Entre las bibliotecas notables se encuentran kaggle, sentence-transformers, PyPDF2, langchain, python-decouple, PyMuPDF, gdown, chromadb, fpdf, SPARQLWrapper, wikidataintegrator, openpyxl, y pycryptodome. Estas bibliotecas proporcionan funcionalidades variadas necesarias para el proyecto.

Por último, debemos importar estas librerías y configurar algunos elementos del entorno, como la supresión de advertencias mediante "`warnings.filterwarnings("ignore")`".

```
!pip install langchain
!pip install python-decouple
!pip install PyMuPDF
!pip install gdown
!pip install chromadb
!pip install fpdf
!pip install SPARQLWrapper
!pip install wikidataintegrator
!pip install openpyxl
!pip install pycryptodome --force

from google.colab import files, drive
import zipfile
import os
from PyPDF2 import PdfReader
import re
import pandas as pd
import numpy as np
import requests
from langchain.text_splitter import RecursiveCharacterTextSplitter
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

Carga del token de Hugging Face:

En este segmento de código, se configuran las variables de entorno necesarias para el proyecto. Se asigna el token de Hugging Face a la variable "`HUGGINGFACE_TOKEN`" utilizando el método config de la biblioteca "`Decouple`". Este token se guarda en un archivo `.env` para asegurar la confidencialidad.

En caso de que no se proporcione un valor en el archivo .env, se utilizará un valor predeterminado. Este enfoque garantiza una gestión segura de información sensible y facilita la flexibilidad en la configuración del proyecto.

Extracción de archivos:

El siguiente código se implementó para facilitar el traspaso de archivos desde una carpeta específica en Google Drive hacia una carpeta local en Google Colab, con solo aclararle la ruta de la misma.

```
# Montar Google Drive en Google Colab
drive.mount('/content/drive')

# Carpeta de origen en Google Drive
source_folder_drive = '/content/drive/MyDrive/Senor de los Anillos'

# Carpeta de destino en Google Colab
destination_folder = '/content/data_pdf'
if not os.path.exists(destination_folder):
    os.makedirs(destination_folder)

# Copiar archivos a la nueva carpeta
for filename in os.listdir(source_folder_drive):
    source_path = os.path.join(source_folder_drive, filename)
    destination_path = os.path.join(destination_folder, filename)
    shutil.copy(source_path, destination_path)

print("Archivos copiados con éxito.")
```

Extracción de texto:

El siguiente paso consistió en la creación de una función “extraer_texto_PDF” para extraer el texto contenido en archivos PDF ubicados en una carpeta específica. El mismo funciona iterando sobre los archivos PDF en la carpeta, extrayendo el texto de cada página y limpiándolo para eliminar los saltos de línea innecesarios.

Luego de esto, se implementó una segunda función llamada “limpiar_saltos”, por medio de la cual se eliminan los saltos de línea que están inmediatamente seguidos por una palabra en un texto dado.

El propósito final de estas funciones es asegurar una presentación más clara del contenido y facilitar su procesamiento posterior.

```

def extraer_texto_PDF(pdf_folder):
    pdf_texts = []

    for pdf_file in os.listdir(pdf_folder):
        if pdf_file.endswith('.pdf'):
            pdf_path = os.path.join(pdf_folder, pdf_file)
            with open(pdf_path, 'rb') as file:
                pdf_reader = PdfReader(file)
                text = ""
                for page_num in range(len(pdf_reader.pages)):
                    page_text = pdf_reader.pages[page_num].extract_text()
                    # Reemplazar saltos de línea simples con espacios y conservar solo dobles
                    cleaned_text = "\n\n".join(para.strip() for para in page_text.split('\n\n'))
                    text += cleaned_text
            pdf_texts.append(text)

    return pdf_texts

# Ruta de la carpeta con los archivos PDF
ruta_PDF = '/content/data_pdf'

# Extraer texto de los PDFs
pdf_texto = extraer_texto_PDF(ruta_PDF)

# Mostrar el texto extraído de un ejemplo de PDF
print(pdf_texto[0][:500]) # Muestra los primeros 500 caracteres del primer PDF

En la adormecida e idílica Comarca, un joven hobbit recibe un encargo:
custodiar el Anillo Único y emprender el viaje para su destrucción
del Destino. Acompañado por magos, hombres, elfos y enanos, atravesará
Tierra Media y se internará en las sombras de Mordor, perseguido siempre

```

Datos tabulares:

Para manejar los datos numéricos tabulares, empleé un pequeño conjunto de datos que contiene información sobre el año de lanzamiento, la duración, la fecha de estreno, presupuesto y recaudación total de la trilogía del “Señor de los Anillos” y “El Hobbit”.

Este conjunto de datos que he creado se encuentra en formato XLSX.

Por último, incorpore una función llamada “*dataframe_to_string*” diseñada para convertir un DataFrame en una cadena de texto formateada. Esta cadena contiene la información del DataFrame en un formato legible para su visualización o almacenamiento.

```

def dataframe_to_string(df):
    result = ""
    for index, row in df.iterrows():
        result += f"Titulo: {row['Titulo']}\n"
        result += f"Fecha de Estreno: {row['Fecha de Estreno']}\n"
        result += f"Duracion: {row['Duracion']}\n"
        result += f"Presupuesto: {row['Presupuesto (USD)']}\n"
        result += f"Recaudacion Total Mundial: {row['Recaudacion total mundial (USD)']}\n\n"

    return result

# Convertir el DataFrame en un string
result_string = dataframe_to_string(df_señor_anillos)

print(result_string)

Titulo: El Señor de los Anillos: La Comunidad del Anillo
Fecha de Estreno: 19 de diciembre de 2001
Duracion: 178 min
Presupuesto: $93 millones
Recaudacion Total Mundial: 871530324

```

Base de datos de grafos:

Para llevar a cabo este proceso elegí utilizar Wikidata como la base de datos de grafos para realizar consultas en línea, ya que la misma, consta de una amplia gama de datos estructurados, modelo de datos flexible y consultas SPARQL. Esta fuente se emplea exclusivamente cuando se formulan preguntas relacionadas con datos de personajes.

Para ello se creó una función llamada “*obtener_valores_por_personaje_tolkien*” la cual realiza una consulta SPARQL a Wikidata para obtener información detallada sobre un personaje específico de las obras de J.R.R. Tolkien.

La consulta busca propiedades de la descripción, país de origen, género, afiliaciones, enemigos y ocupación del personaje.

Si no se encuentra información para el personaje especificado, se devuelve un mensaje indicando que “*No se encontró información*”.

```
# Uso de la función
label_personaje_hobbit_senior_anillos = "Thorin" # Reemplaza con el personaje deseado
print(obtener_valores_por_personaje_tolkien(label_personaje_hobbit_senior_anillos))

Thorin :
Descripción: personaje de El hobbit, de J. R. R. Tolkien, un enano
Reino: Desconocido
Raza: masculino
Miembro de: Compañía de Thorin
Enemigo: Desconocido
Papel: espadachín
```

Base de datos vectorial de embeddings:

Una vez que ya tenemos los PDF en el formato necesario, vamos a realizar la base de datos vectorial con los embeddings.

Para ello se realizaron los siguientes procedimientos:

1. División de los textos:

Este paso consiste en definir el tamaño de los fragmentos de texto como 1000 caracteres (*chunk_size*) y el solapamiento entre fragmentos como 200 caracteres (*chunk_overlap*). Estos valores determinarán cómo se dividen los textos de los PDFs en fragmentos más pequeños.

2. Limpieza de datos:

Se implementó la función “*clean_text_english*” la cual lleva a cabo las siguientes tareas:

- Convertir todo el texto a minúsculas para garantizar consistencia en el procesamiento posterior.
- Eliminar todos los signos de puntuación del texto para simplificar el procesamiento.
- Eliminar todos los números del texto, ya que no son relevantes.
- Conservar símbolos especiales relevantes

- Eliminar stop words en español ya que son palabras comunes y poco informativas.
- Aplicar lematización, es decir, reducirlas a su forma base (lemma) para facilitar la comparación y el análisis.

Luego, aplicamos esta función de limpieza a los documentos divididos y el texto limpio se almacena en una lista llamada “*cleaned_documents*”.

3. Carga del modelo de embeddings:

Utilizamos el modelo pre entrenado "paraphrase-multilingual-mpnet-base-v2" de la biblioteca sentence-transformers para obtener embeddings de oraciones.

Para ello generamos los embeddings de cada documento limpio utilizando el modelo mencionado anteriormente.

Los embeddings capturan la representación semántica de los documentos en un espacio de características de alta dimensión.

4. Almacenar los embeddings en base de datos ChromaDB:

Para este paso empleamos la base de datos ChromaDB para crear una colección llamada "seniordelosanillos_embeddings".

Los embeddings generados se guardan en la colección junto con los fragmentos sin limpiar. Cada documento se asocia con un ID único el cual posteriormente nos va a servir para realizar consultas.

```
# Example query
query_texts = "¿Who is Bilbo Bolson?"
query_embedding = embed_model.encode(query_texts).tolist()

results = collection.query(query_embeddings= [query_embedding], n_results=3)
for result in results['documents'][0]:
    print(result)
    print('\n')

creía que bilbo hubiera muerto cuando le preguntaban dónde está entonces se encogía de hombros vivía sol

cómo lo descubrió preguntó frodo en cuanto al nombre se lo dijo bilbo mismo muy tontamente luego le fue

bilbo se convierte en huésped de elrond se instala en rivendel gandalf visita frodo en la comarca sigue
```

Implementación del RAG:

La implementación de la técnica Retrieval Augmented Generation (RAG) consiste en los siguientes procesos que se detallan a continuación:

Traducción de prompts y respuestas:

En este primer proceso se llevó a cabo la traducción de las preguntas de entrada al inglés para facilitar la interacción con el chatbot, dado que los modelos de lenguaje natural tienden a ser más avanzados en este idioma debido a su entrenamiento.

Utilicé los modelos "opus-mt-es-en" y "opus-mt-en-es" de Helsinki-NLP, disponibles en HuggingFace, para realizar el proceso de traducción.

Es por este motivo que se creó la función "translate_sentence" la cual es la que va a llevar a cabo la traducción del español al inglés.

Por último la función "translate_sentence_to_spanish" se encargará de traducir esa respuesta en inglés al español.

```
# Ejemplo de uso
english_sentence = "Hello, how are you?"
spanish_translation = translate_sentence_to_spanish(english_sentence)

print(f"English Sentence: {english_sentence}")
print(f"Spanish Translation: {spanish_translation}")

English Sentence: Hello, how are you?
Spanish Translation: Hola, ¿cómo estás?
```

Clasificación para elegir la fuente de contexto:

En el proceso de clasificación se creó la función "zephyr_chat_template" para elegir la fuente de contexto. Esta función toma una lista de mensajes, genera un template Jinja para formatear los mensajes y luego devuelve el texto resultante.

Posteriormente implementamos la función "choose_data_source" que utiliza el formato prompt Few-Shot. Consiste en una técnica utilizada en el aprendizaje automático cuya función es proporcionar ejemplos de entrada y salida para guiar al modelo en su proceso de generación de texto. En este caso, el prompt incluye una serie de intercambios de mensajes entre el usuario y el asistente, lo que proporciona ejemplos de interacción típicos.

Estos ejemplos de interacción permiten al modelo comprender el contexto y la intención del usuario, lo que produce una mejor calidad y relevancia de las respuestas generadas por el sistema.

```
# Ejemplo
prompt = '¿Cuál es el papel de Gollum en la trama y cómo afecta el destino del Anillo?'
answer = choose_data_source(prompt, api_key="HUGGINGFACE_TOKEN")
print('\nRESPUESTA: \n', answer)

RESPUESTA:

"Gollum es un personaje en la trilogía cinematográfica de La Comunidad del Anillo. Se trata de un antiguo enano
"En cuanto a cómo afecta el destino del Anillo, Gollum es fundamental para la misión de los hobbits, puesto que
"En resumen, Gollum es un personaje crucial para la trama de La Comunidad del Anillo, ya que conoce la forma de
```

Interpretación de respuesta:

Debido a que le solicité un formato de respuesta específico al LLM, podemos utilizar la función "answer_cleaner" para obtener la información necesaria, esta se formateará en una lista con dos elementos: el primero indicará el tema de la pregunta ("Personaje", "Libros", "Películas"), y el segundo será "None" a menos que se trate de una pregunta sobre un personaje. En ese caso, también se proporcionará el nombre del personaje para realizar la consulta en la base de datos de grafos.

```

def answer_cleaner(texto):
    palabras_clave = ["Personaje", "Libros", "Peliculas"]

    for palabra in palabras_clave:
        if palabra in texto:
            if palabra == "Personaje":
                match = re.search(r'\[(\w*)\]', texto)
                if match:
                    texto_entre_corchetes = match.group(1)
                    return [palabra, texto_entre_corchetes]
            else:
                return [palabra, None]

    return [None, None]

# Ejemplo de uso:
respuesta limpia = answer_cleaner(answer)
print(respuesta limpia)

['Libros', None]

```

Devolución de contexto:

Teniendo en cuenta el resultado de la clasificación, el siguiente paso consiste en transferir la información necesaria a cada fuente para obtener el contexto.

Para ello se implementó la función “*devolver_contexto*” la cual toma el resultado de la función “*answer_cleaner*” (“Personaje”, “Libros”, “Películas”, None) y devuelve el contexto relevante para esa respuesta.

```

# Ejemplo
contexto = devolver_contexto(respuesta limpia, prompt)[1]

print(contexto)

Base de datos vectorial de Embeddings

```

Preguntar al agente con el contexto aclarado:

El último proceso para la implementación de la técnica RAG es preguntarle al LLM sobre un determinado personaje o alguna situación en particular acerca del libro o película.

Es importante recordar que a esa pregunta se le tiene que agregar el contexto y pedirle al LLM que no utilice su conocimiento previo.

Esto lo realizamos con la función “*pregunta_agent_context*” en la cual le especificamos al LLM con un formato de Prompt Few-Shot dos tipos de mensajes:

- Mensaje del sistema: este mensaje indica el rol del sistema, que responde con respuestas verdaderas, útiles y basadas en hechos. Este mensaje proporciona una guía para el modelo sobre el comportamiento esperado del mismo.
- Mensaje del usuario: el mismo proporciona el contexto y la pregunta para el modelo de lenguaje. El contexto se muestra primero, seguido de la pregunta del usuario.

En este formato, se solicita explícitamente al modelo que responda a la pregunta dada, sin utilizar conocimiento previo. Por lo tanto, el modelo debería generar una respuesta basada únicamente en el contexto proporcionado y la pregunta planteada.

```
#Ejemplo
respuesta_final = pregunta_agent_context(prompt, contexto, api_key="HUGGINFACE_TOKEN")

print(f'Prompt: {prompt}\n')
print('-----')
print(f'Contexto: {contexto}\n')
print('-----')
print(f'Respuesta:\n {respuesta_final}')

Prompt: ¿Cuál es el papel de Gollum en la trama y cómo afecta el destino del Anillo?

-----
Contexto: Base de datos vectorial de Embeddings

-----
Respuesta:
Gollum, en la trama de La Comunidad del Anillo de J.R.R. Tolkien, es un personaje dividido en dos personalidades: Gollum, un grotesco
```

Uso de la técnica RAG:

Gracias a todos estos procesos pudimos implementar la función “RAG” la cual realiza los siguientes pasos:

- Traduce el Prompt al inglés.
- Identifica la fuente.
- Formatea una respuesta del LLM para la fuente.
- Obtiene el contexto de la fuente indicada y el ID de la misma.
- Devuelve una respuesta con el prompt original más el contexto obtenido de las fuentes externas.
- Traduce la respuesta al español.

```
def RAG(prompt, api_key):
    #Traducir el prompt al inglés
    english_prompt = translate_sentence(prompt)

    # Identificar la fuente
    source_raw = choose_data_source(english_prompt , api_key="HUGGINFACE_TOKEN")

    # Formatear respuesta del LLM para la fuente
    source_clean = answer_cleaner(source_raw)

    # Obtener el contexto de la fuente indicada y el ID de la fuente
    contexto, id_source = devolver_contexto(source_clean, prompt)

    # Obtener la respuesta con el prompt original más el contexto obtenido de las fuentes externas
    respuesta_final = pregunta_agent_context(prompt, contexto, api_key="HUGGINFACE_TOKEN")

    # Traducir la respuesta al Español
    respuesta_final_español = translate_sentence_to_spanish(respuesta_final)

    return (respuesta_final_español, id_source, contexto)
```

Resultado final:

```
prompt = 'Películas: ¿Cuál es el papel de Thorin en la lucha contra Sauron?'
# Llamar a la función del RAG
respuesta, fuente, contexto = RAG(prompt, "HUGGINGFACE_TOKEN")

# Imprimir el resultado
print('-----')
print(f'PREGUNTA:\n{n(prompt)}')
print('-----')
print(f'FUENTE ESCOGIDA: {fuente}')
print('-----')
print(f'CONTEXTO BRINDADO:\n{n(contexto)}')
print('-----')
print(f'RESPUESTA:\n{n(respuesta)}')

PREGUNTA:
Películas: ¿Cuál es el papel de Thorin en la lucha contra Sauron?
-----
FUENTE ESCOGIDA: Base de datos de grafos
-----
CONTEXTO BRINDADO:
Thorin :
Descripción: personaje de El hobbit, de J. R. R. Tolkien, un enano
Reino: Desconocido
Raza: masculino
Miembro de: Compañía de Thorin
Enemigo: Desconocido
Papel: espadachín
-----
RESPUESTA:
Thorin, como se indica en el contexto, es un personaje de El hobbit de J. R. R. Tolkien, y se describe como un enano.
```

```
prompt = 'Libros: ¿Qué eventos llevan a la formación de la Comunidad del Anillo y cuál es su objetivo principal?'
# Llamar a la función del RAG
respuesta, fuente, contexto = RAG(prompt, "HUGGINGFACE_TOKEN")

# Imprimir el resultado
print('-----')
print(f'PREGUNTA:\n{n(prompt)}')
print('-----')
print(f'FUENTE ESCOGIDA: {fuente}')
print('-----')
print(f'CONTEXTO BRINDADO:\n{n(contexto)}')
print('-----')
print(f'RESPUESTA:\n{n(respuesta)}')

PREGUNTA:
Libros: ¿Qué eventos llevan a la formación de la Comunidad del Anillo y cuál es su objetivo principal?
-----
FUENTE ESCOGIDA: Base de datos vectorial de Embeddings
-----
CONTEXTO BRINDADO:
fuente principal para la historia de la guerra del anillo se llama así por haber sido conservado mucho tiempo en la torres de
-----
RESPUESTA:
Los libros "La Comunidad del Anillo" y "El retorno del Rey" de J.R.R. Tolkien describen la formación y el objetivo principal
```

Conclusión:

Uno de los motivos por los cuales elegí crear este chatbot experto en las trilogías del “Señor de los Anillos” y “El Hobbit” fue con el propósito de facilitarle a los usuarios una respuesta más rápida y completa para todas aquellas preguntas que podrían llegar a tener y que su búsqueda de forma convencional, no solo les llevaría más tiempo sino que existiría la posibilidad de que no haya una respuesta específica para su pregunta.

Por ello, el chatbot tiene la capacidad de interpretar la pregunta del usuario y basándose en el contexto, sin la necesidad de tener que utilizar su conocimiento previo, dar una respuesta específica sobre un determinado personaje o alguna situación en particular acerca del libro o película.

Ejercicio 2: Agentes inteligentes

¿Qué es un agente inteligente impulsado por LLM?

Un agente inteligente impulsado por LLM es un asistente virtual muy avanzado.

Este asistente utiliza programas de computadora especiales que han sido entrenados con una gran cantidad de información sobre cómo funciona el lenguaje en diferentes situaciones, puede traducir textos automáticamente, entender si algo escrito tiene un tono positivo o negativo, e incluso crear textos creativos.

Una forma didáctica de poder entender lo mencionado anteriormente es pensar que estamos ante un robot que entiende lo que le decís y puede responder de una manera inteligente.



¿Por qué se destacan los agentes de LLM?

Los agentes impulsados por modelos de lenguaje de gran escala (LLM) se destacan por varias razones:

- Comprensión contextual: los LLM, como GPT-3, tienen la capacidad de entender el contexto en el que se encuentran las palabras en una oración. Permite comprender mejor el significado de las palabras y generar respuestas más precisas y contextualmente relevantes.
- Generación de texto coherente: estos agentes son capaces de generar texto de manera coherente y relevante. Pueden responder preguntas, completar oraciones y realizar tareas.
- Adaptabilidad a diferentes tareas: los LLM pueden realizar traducciones automáticas, análisis de sentimientos, resúmenes de texto y otras tareas sin necesidad de entrenamiento específico para cada una de ellas.

- Aprendizaje previo en grandes cantidades de datos: estos agentes se entrena n previamente con grandes cantidades de datos textuales, lo que les brinda un conocimiento general del lenguaje. Esto les permite comprender y manejar información en diferentes dominios y estilos de lenguaje.
- Capacidad para manejar preguntas complejas: debido a su comprensión contextual y su capacidad para generar texto de manera coherente, los agentes de LLM pueden manejar preguntas complejas y generar respuestas detalladas.
- Aprendizaje continuo: algunos sistemas basados en LLM permiten el aprendizaje continuo, lo que significa que pueden mejorar su rendimiento con el tiempo a medida que se les proporciona más información y retroalimentación.

Casos de uso para agentes de LLM:

Único agente:

En este caso nos estamos refiriendo a situaciones donde un único agente o entidad es responsable de realizar tareas específicas.

Algunos casos donde se utiliza un único agente son:

- Asistente personal virtual: se trata de un agente de asistencia personal que puede realizar tareas como enviar mensajes, establecer recordatorios, proporcionar información, y realizar otras acciones basadas en comandos de voz o texto.
- Generación de contenido creativo: en este caso estamos ante un agente que crea contenido creativo, como escribir artículos, cuentos o poesía, basándose en parámetros y temas específicos proporcionados por el usuario.



Sistema multiagente:

Este sistema nos permite que los agentes interactúen entre sí de manera colaborativa o competitiva. Un ejemplo podría ser un sistema multiagente colaborativo para traducción automática. Cada agente representa un idioma específico y contribuye a la detección del idioma, segmentación de frases, traducción de segmentos y revisión de calidad, permitiendo una traducción más precisa y coherente. Siguiendo con la explicación de este ejemplo resulta conveniente detallar los agentes y los beneficios de utilizar un sistema multiagente.

Agentes:

- Detección de idioma.
- Segmentación de frases.
- Traducción por pares de idiomas.
- Revisión de calidad.
- Ensamblaje final.

Beneficios:

- Aprovecha la especialización de cada agente.
- Mejora la calidad y coherencia de las traducciones.
- Aborda aspectos específicos del proceso de traducción de manera eficiente.

Cooperación de persona - agente:

Es utilizada en varios ámbitos ya que puede brindar asistencia y colaboración para abordar tareas de manera más eficiente y segura.

Un ejemplo práctico es el de un asistente de escritura colaborativo agente - humano, es decir, se trata de un sistema cooperativo donde un agente de procesamiento del lenguaje natural colabora con un humano para mejorar la calidad y eficiencia de la escritura.

Roles:

Agente de sugerencias:

- Proporciona sugerencias gramaticales, mejora de estilo y sinónimos durante la redacción.

Usuario humano:

- Escribe el texto original y revisa las sugerencias del agente.

Interacción:

- El usuario humano inicia la redacción de un documento.
- El agente sugiere correcciones y mejoras en tiempo real mientras el usuario escribe.
- El usuario revisa y decide aceptar o rechazar cada sugerencia del agente.
- Ambos colaboran en la creación final del documento.

Beneficios:

- Aumenta la calidad del texto con correcciones gramaticales y sugerencias de estilos.

- Mejora la eficiencia del proceso de escritura al proporcionar sugerencias instantáneas.
- Facilita la colaboración entre la inteligencia artificial y la creatividad humana.



Investigación sobre las aplicaciones que actualmente ofrecen agentes de IA para distintos objetivos:

AutoGen es un proyecto de Microsoft Research que permite a los desarrolladores construir equipos de agentes de IA multi-tarea para lograr casi cualquier objetivo. Estos agentes de IA pueden escribir y ejecutar códigos, criticarse mutuamente, planear, adoptar diferentes roles y personalidades, entre otras cosas.

Lo que hace único a AutoGen es su capacidad para ejecutar iteraciones automatizadas y ejecutar código en un entorno contenedorizado. Con AutoGen o cualquier marco multi-agente, se puede habilitar a otra IA para proporcionar comentarios, acortando el ciclo de los mismos y permitiendo menos trabajo para el usuario humano.

Además, AutoGen es un entorno que simplifica la orquestación, optimización y automatización de flujos de trabajos LLM. Ofrece agentes personalizables y conversables que aprovechan las capacidades más potentes de los LLM más avanzados, como GPT-4, al tiempo que abordan sus limitaciones mediante la integración con humanos y herramientas y la celebración de conversaciones entre varios agentes a través del chat automatizado.

Otros ejemplos de aplicaciones son:

- AutoGPT: por medio de esta aplicación se produce un cambio drástico en la relación entre la inteligencia artificial y el usuario final. Mientras que ChatGPT se basa en un intercambio entre la IA y el usuario, donde le das una solicitud y la IA devuelve un resultado, Auto-GPT solo necesita un mensaje inicial de tu parte. A partir de ahí, el agente de IA generará una lista de tareas que cree que necesita para cumplir con lo que le pediste, sin requerir más información o mensajes.
- BabyAGI: este proyecto se centra en el desarrollo de agentes autónomos con habilidades de planificación a largo plazo y uso de memoria.

- CAMEL and Generative Agents: los mismos han ganado popularidad y han sido implementados en la comunidad LangChain.
- AGENTS: se trata de un nuevo framework *open source* por medio del cual facilita la creación y personalización de estos agentes.

Problemática a solucionar con un sistema multiagente:

La problemática que se propone solucionar es la de la traducción automática en tiempo real en una plataforma de chat multilingüe. En la misma, usuarios de diferentes lenguajes pueden interactuar entre sí en tiempo real, con sus mensajes traducidos automáticamente a su lenguaje nativo.

El sistema multiagente consta de tres tipos de agentes:

1. Agente de usuario: este agente interactúa directamente con el usuario, recibe el mensaje en su lenguaje nativo y lo envía al agente de traducción.
2. Agente de traducción: es quien recibe el mensaje del agente de usuario, lo traduce al lenguaje deseado, utilizando un modelo de traducción automática, y luego envía el mensaje traducido al agente de destino.
3. Agente de destino: el mismo recibe el mensaje traducido del agente de traducción y lo presenta al usuario destinatario en su lenguaje nativo.

Interacción entre los agentes:

Un ejemplo podría ser el siguiente:

- El usuario Alejo (habla español) le escribe un mensaje a su agente de usuario: “*Hola, ¿cómo estás?*”
- El agente de usuario de Alejo le envía al agente de traducción el mensaje recibido: {“*mensaje*”: “*Hola, ¿cómo estás?*”, “*lenguaje origen*”: “*español*”, “*lenguaje destino*”: “*inglés*”}
- El agente de traducción le envía al agente de destino de Ernestina: {“*mensaje traducido*”: “*Hello, how are you?*”}
- Por último el agente de destino de Ernestina le envía el mensaje a la destinataria, Ernestina (habla inglés): “*Hello, how are you?*”



Esquema del Sistema Multiagente:

El usuario Alejo (español)



El agente de usuario de Alejo: recibe el mensaje del usuario en su lenguaje nativo.



El agente de traducción: recibe el mensaje del agente de usuario de Alejo y procede a traducirlo al lenguaje deseado.



El agente de destino de Ernestina: recibe el mensaje traducido del agente de traducción.



El usuario Ernestina (inglés)

En este esquema, cada flecha representa la dirección del flujo de información. Cuando el usuario Alejo envía un mensaje, el mismo es recibido por su agente de usuario. Luego éste le envía el mensaje al agente de traducción, que se encarga de traducir el

mensaje al lenguaje deseado. Finalmente, el agente de destino de Ernestina recibe el mensaje traducido del agente de traducción y lo presenta al usuario Ernestina.

Informe de la investigación:

El objetivo de utilizar un sistema multiagente para la traducción automática en tiempo real en una plataforma de chat multilingüe, puede facilitar la comunicación entre usuarios que hablan diferentes idiomas. Sin embargo, la eficacia del sistema depende en gran medida de la precisión de los modelos de traducción automática utilizados.

Es importante destacar que este esquema es una simplificación del sistema real.

En un sistema real, puede haber múltiples agentes de cada tipo, y estos pueden interactuar entre sí de formas más complejas, ya que pueden tener su propia inteligencia y capacidad para tomar decisiones. Sin embargo, puede afectar la forma en la que se realiza la traducción y se presenta la información al usuario. Por ejemplo, el agente de traducción puede tener la capacidad de elegir entre diferentes modelos de traducción basándose en el contexto del mensaje y el idioma deseado. Asimismo, el agente de usuario y el agente de destino pueden tener la capacidad de personalizar la forma en que presentan la información al usuario basándose en las preferencias y el comportamiento de este.

Bibliografía:

1. Avila, D. (20 de abril de 2023). *Agentes Autónomos y Simulaciones en LLM: Un vistazo a AutoGPT, BabyAGI, CAMEL y Generative Agents*. Medium. Recuperado de [Agentes Autónomos y Simulaciones en LLM: Un vistazo a AutoGPT, BabyAGI, CAMEL y Generative Agents.](#)
2. Dimensiona. (24 de octubre de 2023). *AutoGen: permite aplicaciones de IA de próxima generación mediante conversaciones de agentes múltiples*. DimensionIA. Recuperado de [AutoGen: permite aplicaciones de IA de próxima generación mediante conversaciones de agentes múltiples.](#)
3. Dattaraj, R. (30 de junio de 2023). *Cómo los modelos de lenguaje grande (LLM) impulsarán las aplicaciones del futuro*. Unite AI. Recuperado de [Cómo los modelos de lenguaje grande \(LLM\) impulsarán las aplicaciones del futuro.](#)
4. Fernandez Sanches, J. (12 de junio de 2023). *Estado del arte de las IAs: la inteligencia artificial en la actualidad*. Hiberus Blog. Recuperado de [Estado del arte de las IAs: la inteligencia artificial en la actualidad.](#)
5. *Modelos de lenguaje grande (LLM): guía completa en 2023*. Shaip. Recuperado de [Modelos de lenguaje grande \(LLM\): guía completa en 2023.](#)