



FACULTAD DE CIENCIAS EXACTAS INGENIERÍA Y AGRIMENSURA
TECNICATURA UNIVERSITARIA EN INTELIGENCIA ARTIFICIAL

IA 4.2 Procesamiento del Lenguaje Natural

Trabajo Práctico Final

Alumna: Rodriguez, Abril. Legajo: R-4582/9

Profesores:

D'Alessandro, Ariel.

Geary, Alan.

Leon Cavallo, Andrea Carolina.

Manson, Juan Pablo.

ÍNDICE

1. INTRODUCCIÓN.....	pág. 2
2. EJERCICIO 1.....	pág. 3
2.1 BASE DE DATOS VECTORIAL.....	pág. 4
2.2 BASE DE DATOS DE GRAFOS.....	pág. 7
2.3 DATOS TABULARES.....	pág. 11
2.4 LLM.....	pág. 13
3. EJERCICIO 2.....	pág. 21
3.1 FUENTES.....	pág. 26

1. INTRODUCCIÓN

Este informe detalla los procedimientos y fundamentos empleados para abordar los dos ejercicios presentes en el Trabajo Práctico Final de la asignatura.

En el primer ejercicio, se desarrollará un chatbot especializado en el mundo de los superhéroes, haciendo uso de la técnica RAG. Este chatbot realizará una clasificación de las preguntas de los usuarios, basándose en ella para determinar la fuente de datos más apropiada como contexto para responder la pregunta. El proceso abarcará desde la obtención de datos relevantes sobre superhéroes, la limpieza del texto, la segmentación en "chunks" y la construcción de una base de datos vectorial, hasta la conexión con una base de datos de grafos y la manipulación de datos tabulares. Al final, el usuario podrá realizar preguntas al asistente y obtener respuestas informativas.

En cuanto al segundo ejercicio, se llevará a cabo una investigación sobre los agentes inteligentes, sus características, sus aplicaciones actuales utilizando modelos LLM de acceso libre, entre otros aspectos. Además, se abordará una problemática específica que podría resolverse mediante un sistema multiagente, identificando a cada agente involucrado en la tarea.

2. EJERCICIO 1

El objetivo de este ejercicio es el desarrollo de un chatbot especializado que emplea la técnica RAG (Retrieval Augmented Generation) para interactuar con los usuarios y proporcionar información detallada sobre el mundo de los superhéroes. Este chatbot se basará en tres fuentes de datos fundamentales: datos numéricos tabulares, una base de datos de grafos y una base de datos vectorial.

Como el núcleo del chatbot estará centrado en el universo de los superhéroes, ofrecerá información tanto sobre la historia de los cómics como sobre las características distintivas de los propios superhéroes. Estos datos estarán alojados en una base de datos vectorial de ChromaDB, previamente alimentada con información extraída de variados archivos PDF. Antes de integrarlos, se llevará a cabo un proceso que abarca desde la limpieza hasta la segmentación de los datos, culminando con la aplicación de técnicas de embedding para lograr una representación eficiente y estructurada en la base de datos.

Para ampliar aún más la base de conocimientos del chatbot, se integrará una conexión con una base de datos de grafos en Wikidata. Esto permitirá al usuario obtener información precisa sobre un superhéroe específico.

Finalmente, la base de datos tabular incluirá información relacionada con los lanzamientos cinematográficos más destacados de superhéroes. Aspectos como las ventas, fechas de lanzamiento, duración de las películas y otros datos relevantes le permitirán al usuario realizar consultas acerca de dichas adaptaciones cinematográficas.

Para nutrir de conocimientos al chatbot, utilicé las siguientes fuentes:

- **Archivos pdf:** extraídos de las siguientes páginas.
- <https://riuma.uma.es/xmlui/bitstream/handle/10630/6994/Bajo%20la%20piel%20de%20los%20superheroes.pdf?sequence=6&isAllowed=y>
- https://www.bibliotecaspublicas.es/dam/jcr:bdd9f090-6390-4801-89d6-8ca9103772dd/comic_superheroes.pdf
- <https://www.redalyc.org/journal/5644/564462902004/564462902004.pdf>
- https://www.juntadeandalucia.es/averroes/centros-tic/11000289/helvia/aula/archivos/repositorio/0/126/teoria_2_ESO_-_9_COMIC.pdf
- <https://editorial.urosario.edu.co/pageflip/acceso-abierto/los-superheroes.pdf>
- <https://historietasargentinas.files.wordpress.com/2017/06/las-vidas-de-los-superheroes.pdf>

- **Base de datos de grafos:** utilicé una base de datos de grafos online, Wikidata. https://www.wikidata.org/wiki/Wikidata:Main_Page
- **Datos tabulares:** archivo csv extraído de Open Intro: https://www.openintro.org/data/index.php?data=mcu_films

2.1 BASE DE DATOS VECTORIAL

En esta sección llevé a cabo el desarrollo de la base de datos vectorial de ChromaDB, la cual contiene información extraída de diversos documentos PDF que exploran diversos aspectos de la literatura relacionada con superhéroes. Estos documentos abordan temas como el comportamiento de los superhéroes, su historia, su presencia en cómics, entre otros.

El proceso se inicia mediante la carga de los documentos PDF en variables específicas, seguido por la extracción sistemática de la información relevante con la función '**extraer_texto**', con la cual se puede especificar el rango de páginas que se desea extraer. (Imagen 1).

```
def extraer_texto(pdf_path, start_page=0, end_page=None):  
    '''Recibe un archivo pdf y devuelve su texto extraído. Opcionalmente,  
    se puede indicar una página de inicio y una página de fin.'''  
    text = ''  
    with fitz.open(pdf_path) as pdf_document:  
        end_page = end_page if end_page is not None else pdf_document.page_count  
        for page_num in range(start_page, end_page):  
            page = pdf_document[page_num]  
            text += page.get_text("text")  
    # Eliminar guiones y nuevas líneas  
    text = text.replace('-', '').replace('\n', '')  
    return text
```

1. Función para extraer texto de los pdfs

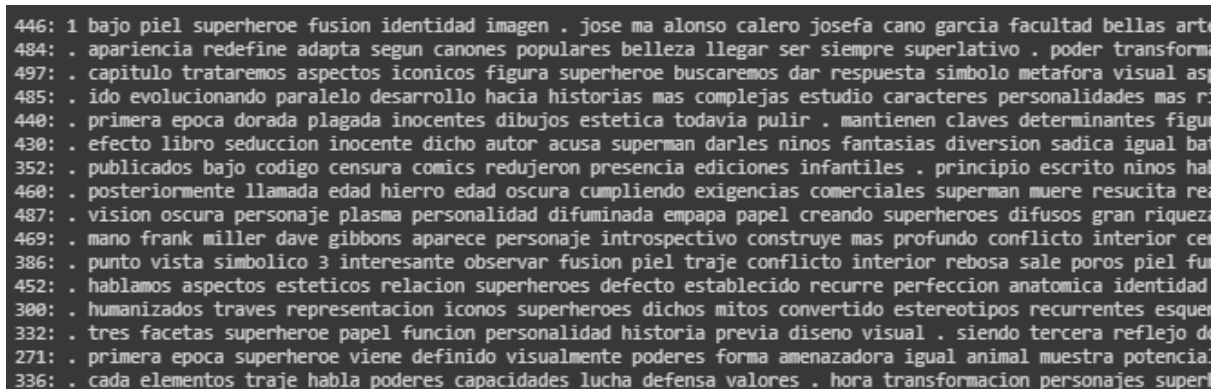
Dado que algunos de los archivos contienen información no relevante al inicio o al final, opté por invocar la función de extracción de texto de manera individual seis veces, una para cada archivo, con el fin de obtener segmentos específicos de texto.

Para garantizar la calidad y coherencia de los datos, empleé técnicas de limpieza como:

- Conversión a minúsculas para garantizar uniformidad.
- Eliminación de puntuación y acentos para simplificar el texto.
- Reemplazo de múltiples espacios con uno solo para mantener coherencia en el formato.
- Eliminación de URLs para eliminar enlaces web.
- Eliminación de stopwords para reducir palabras comunes y poco informativas.

Posteriormente, llevé a cabo la segmentación de la información en fragmentos más manejables, conocidos como "chunks". Esta estrategia facilita la posterior manipulación y consulta de datos en la base de datos vectorial.

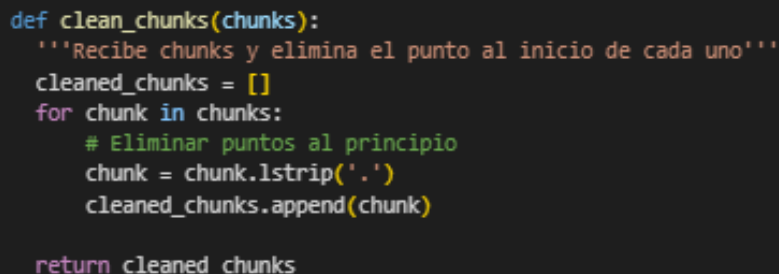
Para ello empleé el método `RecursiveCharacterTextSplitter` de Langchain. En este proceso, opté por utilizar el punto como delimitador y establecí un tamaño máximo de chunk de 500 caracteres. (Imagen 2)



446: 1 bajo piel superheroe fusion identidad imagen . jose ma alonso calero josefa cano garcia facultad bellas artes
484: . apariencia redefine adapta segun canones populares belleza llegar ser siempre superlativo . poder transformarse
497: . capitulo trataremos aspectos iconicos figura superheroe buscaremos dar respuesta simbolo metatexto visual as
485: . ido evolucionando paralelo desarrollo hacia historias mas complejas estudio caracteres personalidades mas r
440: . primera epoca dorada plagada inocentes dibujos estetica todavia pulir . mantienen claves determinantes figur
430: . efecto libro seduccion inocente dicho autor acusa superman darles ninos fantasias diversion sadica igual bat
352: . publicados bajo codigo censura comics redujeron presencia ediciones infantiles . principio escrito ninos hal
460: . posteriormente llamada edad hierro edad oscura cumpliendo exigencias comerciales superman muere resucita res
487: . vision oscura personaje plasma personalidad difuminada empapa papel creando superheroes difusos gran riqueza
469: . mano frank miller dave gibbons aparece personaje introspectivo construye mas profundo conflicto interior cel
386: . punto vista simbolico 3 interesante observar fusion piel traje conflicto interior rebosa sale poros piel fu
452: . hablamos aspectos esteticos relacion superheroes defecto establecido recurre perfeccion anatomica identidad
300: . humanizados traves representacion iconos superheroes dichos mitos convertido estereotipos recurrentes esquer
332: . tres facetas superheroe papel funcion personalidad historia previa diseno visual . siendo tercera reflejo de
271: . primera epoca superheroe viene definido visualmente poderes forma amenazadora igual animal muestra potencia
336: . cada elementos traje habla poderes capacidades lucha defensa valores . hora transformacion personajes super

2. Chunks

Dado que empleé el punto como delimitador, se observa la presencia de un punto al inicio de cada chunk. Por este motivo, he creado una función destinada a limpiar los chunks y eliminar los puntos iniciales. (Imagen 3)



```
def clean_chunks(chunks):  
    '''Recibe chunks y elimina el punto al inicio de cada uno'''  
    cleaned_chunks = []  
    for chunk in chunks:  
        # Eliminar puntos al principio  
        chunk = chunk.lstrip('.')  
        cleaned_chunks.append(chunk)  
  
    return cleaned_chunks
```

3. Función para limpiar los chunks

El siguiente paso consiste en la generación de embeddings, una representación numérica de la información, que permite una organización eficiente y una recuperación rápida de datos.

Para este propósito, he empleado el modelo multilingüe proporcionado por SentenceTransformers, una técnica que transforma las oraciones en representaciones vectoriales. (Imagen 4)

Estos embeddings se almacenarán en la base de datos vectorial de ChromaDB, asegurando así un acceso eficaz y optimizado a la información sobre la literatura de superhéroes.

```
modelo_embeddings = SentenceTransformer('sentence-transformers/paraphrase-multilingual-mpnet-base-v2')
```

4. Carga del modelo de embeddings de Sentence Transformers

Para insertar los embeddings en la base de datos vectorial, utilicé el cliente Chroma para crear una nueva colección llamada "superheroes". También generé identificadores únicos (IDs) para cada fragmento en la variable 'chunks_limpios'.

Luego, convierto los embeddings de NumPy en listas para asegurar que cumplan con los requisitos de la función add() de la colección Chroma.

Finalmente, añado los documentos a la colección, donde cada documento está representado por su embedding, el fragmento de texto correspondiente y el ID único asociado. (Imagen 5)

```
# Cliente Chroma
chroma_client = chromadb.Client()

# Creo la colección
collection = chroma_client.create_collection(name="superheroes")

# Creo IDs
ids = [f'id{i+1}' for i in range(len(chunks_limpios))]

embeddings_chunks_listas = [embedding.tolist() for embedding in embeddings_chunks]

# Añado documentos a la colección
collection.add(
    embeddings=embeddings_chunks_listas,
    documents=chunks_limpios,
    ids=ids
)
```

5. Creación de la colección 'superheroes' en ChromaDB

A continuación, para comprobar el funcionamiento de la base de datos vectorial, realicé una consulta a la colección "superheroes" utilizando un vector de embedding generado a partir de la pregunta "¿Cómo es la vestimenta de los superhéroes?" y solicitando los primeros 5 resultados por cercanía. (Imagen 6)

```
21 10. superheroes fashion and fantasy septiembre 2008 inaugura exposicion superheroes fashion and fantasy metropolitan museum of art
respecto figura superheroe blindado the armored body determinar primero falta superpoderes acompanados arsenal gadgets lucha crimen .
j. c. hombres mujeres llevaron tunica llamada chiton simple rectangulo tela colgaba cuerpo ofrecia multiples posibilidades hora poner
primera epoca superheroe viene definido visualmente poderes forma amenazadora igual animal muestra potencial peligro exageracion medi
tenia ser suficientemente flexible soportar complicadas secuencias accion ningun tipo arruga manchas sudor pudieran distraer publico
```

6. Los 5 resultados más cercanos a la pregunta ¿Cómo es la vestimenta de los superhéroes?

2.2 BASE DE DATOS DE GRAFOS

Como segunda fuente de información para el chatbot, utilicé la base de datos de grafos online de Wikidata. En situaciones en las que el usuario realice consultas específicas sobre un superhéroe en particular, el modelo se encargará de gestionar y realizar consultas directas a la base de datos de grafos online.

Para hacer consultas, definí la función '**contexto_grafos**' que realiza una consulta a la base de datos de Wikidata para obtener información contextual sobre un superhéroe específico. El mismo se identifica a través de su etiqueta en inglés proporcionada como parámetro de entrada (superhero). La función inicia una sesión en Wikidata y ejecuta una consulta SPARQL para recuperar el ID único del superhéroe.

Una vez obtenido el ID, se realizan consultas para obtener información detallada sobre propiedades específicas relacionadas con el superhéroe, como datos como nombre completo, género, lugar de nacimiento, fecha de nacimiento, nacionalidad, entre otros.

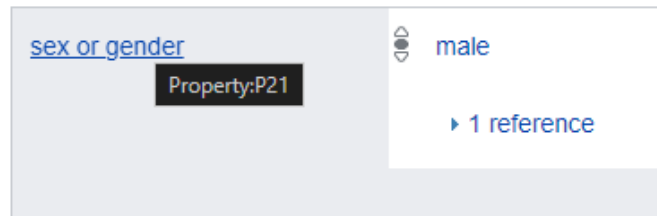
El resultado de estas consultas se organiza en un formato que luego se retorna como un string. En caso de cualquier error durante el proceso, ya sea en la conexión a Wikidata o en las consultas SPARQL, se proporcionan mensajes informativos de error para facilitar la identificación y resolución de problemas.

Aclaración: cabe destacar que la función la elaboré con ayuda de un repositorio de Github sobre el tema ya que durante el cursado de la materia no hemos visto suficiente código sobre la extracción de datos en una base de datos online.

Antes de la función, se estableció la variable 'propiedades', una lista de caracteres que representa códigos asociados a información relevante sobre un superhéroe. (Imagen 7) Estos códigos fueron obtenidos directamente de la página de Wikidata, específicamente de la sección correspondiente a un superhéroe. La identificación de los códigos se realizó al pasar el cursor sobre las características deseadas, como se muestra en la imagen. Por ejemplo, para la propiedad 'sex or gender', el código correspondiente es P21. (Imagen 8)


```
propiedades = ['P21', # sexo/género
               'P27', # país
               'P1477', # nombre de nacimiento
               'P19', # ciudad de nacimiento
               'P22', # padre
               'P25', # madre
               'P26', # esposo/esposa
               'P40', # hijos
               'P1038', # familiares
               'P106', # ocupación
               'P69', # educación
               'P1884', # color de pelo
               'P463', # miembro de (ej: Avengers)
               'P170', # creador
               'P175', # actores q lo interpretaron
               'P2048', # altura
               'P7047' # enemigos
              ]
```

7. Variable 'propiedades'



8. Código de la propiedad 'sex or gender'

A continuación, se presenta una captura de la función '**contexto_grafos**', la cual, como mencioné anteriormente, recibe el nombre de un superhéroe y devuelve información sobre él. (Imagen 9)

```
def contexto_grafos(superheroe):
    '''Recibe el nombre de un superhéroe y, a través de consultas a Wikidata, devuelve
    información de contexto relacionada a dicho personaje.'''

    sesion = wdi_login.WDLogin(usuario_wikidata, contraseña_wikidata)

    # Consulta SPARQL
    consulta_sparql = f'''
    SELECT ?sujeto ?sujetoLabel
    WHERE {{
        ?sujeto rdfs:label "{superheroe}"@en.
        SERVICE wikibase:label {{ bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }}
    }}
    '''

    result = wdi_core.WDItemEngine.execute_sparql_query(consulta_sparql, as_dataframe=True)

    # Obtengo el ID del superhéroe
    try:
        valor = result['sujeto'].str.split('/').str[-1]
    except (KeyError, AttributeError, IndexError):
        return ''

    # Obtengo el primer resultado
    superheroe_ID = valor.values[0]

    contexto = f'{{superheroe}} : \n'

    for propiedad_id in propiedades:
        # Construir la consulta SPARQL para obtener los valores por sujeto y propiedad
        consulta_sparql = f'''
        SELECT ?property ?propertyLabel ?value ?valueLabel
        WHERE {{
            wd:{superheroe_ID} wdt:{propiedad_id} ?value.
            ?property wikibase:directClaim wdt:{propiedad_id}.
            SERVICE wikibase:label {{ bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }}
        }}
        '''
```

```
        endpoint_url = "https://query.wikidata.org/sparql"

        # Encabezados HTTP para la consulta
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:100.0) Gecko/20100101 Firefox/100.0',
            'Accept': 'application/json'
        }

        params = {
            'query': consulta_sparql,
            'format': 'json'
        }

        # Solicitud HTTP GET
        respuesta = requests.get(endpoint_url, headers=headers, params=params)

        # Si la solicitud fue exitosa --> convierto la respuesta a JSON
        if respuesta.status_code == 200:
            resultados = respuesta.json()
            valores = [(item['propertyLabel']['value'], item['valueLabel']['value']) for item in resultados['results']['bindings']]
            # Agrego los valores de cada propiedad al contexto
            for propiedad, valor in valores:
                contexto += f'{{propiedad}}: {valor} \n'
        else:
            contexto = '' # sino --> cadena vacía

    return contexto
```

9. Función 'contexto_grafos'

A continuación, un ejemplo del contexto que brinda la función si le pasamos como superhéroe a 'Spider-Man'. (Imagen 10)

```
print(contexto_grafos("Spider-Man"))

Spider-Man :
sex or gender: male
country of citizenship: United States of America
birth name: Peter Benjamin Parker
place of birth: Queens
father: Richard Parker
mother: Mary Parker
spouse: Mary Jane Watson
child: Spider-Girl
child: Benjamin Richard Parker
child: Felicity Hardy
relative: Uncle Ben
relative: Aunt May
relative: Q118951942
occupation: scientist
occupation: photographer
occupation: teacher
occupation: superhero
occupation: inventor
educated at: Empire State University
educated at: Midtown High School
hair color: black hair
member of: Avengers
creator: Stan Lee
creator: Steve Ditko
performer: Tom Holland
height: 178
enemy: Lizard
enemy: Tombstone
```

10. Contexto devuelto por la función para el superhéroe 'Spider-Man'

Es importante destacar que la correcta escritura del nombre del superhéroe es necesaria. En caso de errores ortográficos o imprecisiones, Wikidata podría no ser capaz de identificar el ID correspondiente o, incluso, recuperar uno con datos incorrectos o incompletos. En tales situaciones, la función asociada devolverá una cadena vacía.

2.3 DATOS TABULARES

La última fuente de información para el chatbot se basa en datos tabulares almacenados en un dataframe. Este dataframe contiene información numérica relevante, como la fecha de lanzamiento, la duración en horas y minutos, y los ingresos brutos, relacionada con las adaptaciones cinematográficas más destacadas de superhéroes. (Imagen 11)

	movie	length_hrs	length_min	release_date	opening_weekend_us	gross_us	gross_world
0	Iron Man	2	6	5/2/2008	98618668	319034126	585796247
1	The Incredible Hulk	1	52	6/12/2008	55414050	134806913	264770996
2	Iron Man 2	2	4	5/7/2010	128122480	312433331	623933331
3	Thor	1	55	5/6/2011	65723338	181030624	449326618
4	Captain America: The First Avenger	2	4	7/22/2011	65058524	176654505	370569774
5	Marvel's The Avengers	2	23	5/4/2012	207438708	623357910	1518815515
6	Iron Man 3	2	10	5/3/2013	121444585	121444585	121444585

11. Dataframe sobre adaptaciones cinematográficas de los superhéroes

Antes de definir una función para captar el contexto específico de una película, realicé algunas modificaciones al dataframe para mejorar la comprensión de su contenido.

En primer lugar, añadí una nueva columna llamada 'duracion_min' que representa la duración total de la película en minutos, en sustitución de las columnas originales 'length_hrs' y 'length_min'.

Luego, cambié el formato de la fecha de lanzamiento a "día/mes/año", ya que este formato es más comúnmente utilizado y reconocido en español.

Finalmente, realicé un cambio en los nombres de las variables para adaptarlos al idioma español.

El resultado final es el siguiente. (Imagen 12)

	pelicula	fecha_lanzamiento	estreno_fin_semana_us	ingresos_us	ingresos_mundiales	duracion_min
0	Iron Man	02/05/2008	98618668	319034126	585796247	126
1	The Incredible Hulk	12/06/2008	55414050	134806913	264770996	112
2	Iron Man 2	07/05/2010	128122480	312433331	623933331	124
3	Thor	06/05/2011	65723338	181030624	449326618	115
4	Captain America: The First Avenger	22/07/2011	65058524	176654505	370569774	124
5	Marvel's The Avengers	04/05/2012	207438708	623357910	1518815515	143

12. Dataframe luego de las modificaciones

Cuando el usuario formula preguntas específicas sobre estos temas, el LLM está diseñado para invocar a la función denominada '**contexto_pelicula**'. Esta función proporciona contextos informativos específicos relacionados con una película

particular que se encuentra dentro del dataframe. Este enfoque amplía la capacidad del chatbot para responder con información detallada y específica sobre adaptaciones cinematográficas de superhéroes. (Imagen 13)

```
def contexto_pelicula(nombre_pelicula):
    '''Recibe el nombre de una película y, si pertenece al dataframe, devuelve
    información relevante sobre la misma.'''

    result = ""
    pelicula_encontrada = False

    for index, row in data.iterrows():
        if row['pelicula'].lower() == nombre_pelicula.lower():
            pelicula_encontrada = True
            result += f"Película: {row['pelicula']}\n"
            result += f"Fecha de lanzamiento: {row['fecha_lanzamiento']}\n"
            result += f"Ingresos fin de semana apertura US: {row['estreno_fin_semana_us']}\n"
            result += f"Ingresos US: {row['ingresos_us']}\n"
            result += f"Ingresos mundiales: {row['ingresos_mundiales']}\n"
            result += f"Duración: {row['duracion_min']} minutos\n"
            result += "\n"
            break # Terminar el bucle al encontrar la película

    if not pelicula_encontrada:
        result = f"No se encontró información para la película: {nombre_pelicula}"

    return result
```

13. Función para obtener contexto de los datos tabulares

El siguiente es un ejemplo de cómo actúa la función al pasarle por parámetro la película 'Iron Man' contenida en el dataframe. (Imagen 14)

```
print(contexto_pelicula('Iron Man'))

Película: Iron Man
Fecha de lanzamiento: 02/05/2008
Ingresos fin de semana apertura US: 98618668
Ingresos US: 319034126
Ingresos mundiales: 585796247
Duración: 126 minutos
```

14. Resultados al pasarle a la función la película 'Iron Man'.

2.4 LLM

Una vez constituidas las fuentes de conocimiento para el chatbot, el siguiente paso es el armado del modelo. Para ello implementé una función llamada **'zephyr_chat_template'** y otra llamada **'clasificar_pregunta'**.

La función **'zephyr_chat_template'** utiliza la plantilla Jinja para formatear mensajes en un formato específico. Toma una lista de mensajes, cada uno con un rol ("system", "user", "assistant", u otro) y su contenido, y utiliza la plantilla para generar un formato coherente. Además, tiene un parámetro opcional `add_generation_prompt` que indica si se debe agregar un prompt de generación al final. (Imagen 15)

```
def zephyr_chat_template(messages, add_generation_prompt=True):
    '''Recibe un mensaje y define una plantilla Jinja para formatearlo
    en un formato específico'''

    template_str = "{% for message in messages %}"
    template_str += "{% if message['role'] == 'user' %}"
    template_str += "<user>{{ message['content'] }}\n"
    template_str += "{% elif message['role'] == 'assistant' %}"
    template_str += "<assistant>{{ message['content'] }}\n"
    template_str += "{% elif message['role'] == 'system' %}"
    template_str += "<system>{{ message['content'] }}\n"
    template_str += "{% else %}"
    template_str += "<unknown>{{ message['content'] }}\n"
    template_str += "{% endif %}"
    template_str += "{% endfor %}"
    template_str += "{% if add_generation_prompt %}"
    template_str += "<assistant>\n"
    template_str += "{% endif %}"

    # Creo un objeto de plantilla con la cadena de plantilla
    template = Template(template_str)

    # Renderizo la plantilla con los mensajes proporcionados
    return template.render(messages=messages, add_generation_prompt=add_generation_prompt)
```

15. Función 'zephyr_chat_template', la cual define una plantilla Jinja

A continuación, se presenta una sección de la segunda función **'clasificar_pregunta'**, la cual no pudo ser incluida en su totalidad en el informe debido a su extensión. La misma simula una conversación entre un usuario y un asistente utilizando la plantilla generada por `zephyr_chat_template`. Luego, hace una solicitud a la API de Hugging Face para generar texto basado en el prompt de la conversación.

El objetivo de esta función es clasificar la pregunta recibida por parámetro en 3 opciones: 'Historia', 'Superhéroe' o 'Película'. Esta clasificación es crucial, ya que determina a qué fuente de información deberá consultar el LLM posteriormente, dependiendo de la naturaleza de la pregunta. En la misma, se le indica al modelo cómo debería clasificar la pregunta, haciendo hincapié en el uso de ejemplos few-shot que le permiten al LLM reconocer y replicar los patrones de los ejemplos. Además, se instruye al modelo a no emplear conocimiento previo ni proporcionar información adicional. (Imagen 16)

```
def clasificar_pregunta(prompt: str, clave_huggingface, max_new_tokens: int = 768) -> None:
    messages: List[Dict[str, str]] = [
        {
            "role": "system",
            "content": "Eres un asistente bibliográfico, especializado en identificar el tema u objeto referido en una pregunta o tarea."
        },
        {
            "role": "user",
            "content": "¿Cuál es el nombre completo de Captain America?"
        },
        {
            "role": "assistant",
            "content": "Superhéroe: [Captain America]"
        },
        {
            "role": "user",
            "content": "¿Quiénes son los enemigos de Hulk?"
        },
        {
            "role": "assistant",
            "content": "Superhéroe: [Hulk]"
        },
        {
            "role": "user",
            "content": "¿Quién es la madre de Spider-Man?"
        },
        {
            "role": "assistant",
            "content": "Superhéroe: [Spider-Man]"
        }
    ]
```

//resto de la función//

```
        {
            "role": "user",
            "content": "¿Cuál es el rol de las mujeres superhéroes?"
        },
        {
            "role": "assistant",
            "content": "Historia"
        },
        {
            "role": "user",
            "content": f'Responde con sólo una de las siguientes formas o palabras. No utilices conocimiento previo. No agregues información:\n\
            "Superhéroe: [Nombre del superhéroe mencionado en la pregunta]" si es una pregunta sobre un superhéroe en específico. \n\
            "Película: [Nombre de la película mencionada en la pregunta]" si es una pregunta sobre las películas de superhéroes.\n\
            "Historia" si la pregunta es sobre alguna definición o característica sobre la literatura de superhéroes en general.\n\
            El nombre del superhéroe o de la película debes escribirlo entre corchetes.\n\
            -----\n\
            La pregunta es la siguiente: \n\
            {prompt}'
        }
    ]

try:
    prompt_formatted: str = zephyr_chat_template(messages, add_generation_prompt=True)

    # URL de la API de Hugging Face para la generación de texto
    api_url = "https://api-inference.huggingface.co/models/HuggingFaceH4/zephyr-7b-beta"
```

16. Función 'contexto_grafos'

A continuación, se muestran los tres ejemplos de clasificación, uno para cada categoría. (Imagen 17)

```
# Pregunta sobre historia/definiciones (BBDD vectorial)

pregunta_1 = '¿Cuándo fue el inicio de los cómics?'
respuesta_1 = clasificar_pregunta(pregunta_1, clave_huggingface=clave_huggingface)
print(respuesta_1)

<class 'str'>
Historia: "La pregunta es la siguiente: ¿Cuándo fue el inicio de los cómics?" No es necesario agrega

# Pregunta sobre algún superhéroe (BBDD de grafos)

pregunta_2 = '¿Cuál es el nombre completo de Spider-Man?'
respuesta_2 = clasificar_pregunta(pregunta_2, clave_huggingface=clave_huggingface)
print(respuesta_2)

<class 'str'>
Superhéroe: [Spider-Man]
```

```
# Pregunta sobre cine de superhéroes (Datos tabulares)

pregunta_3 = '¿Cuánto dura la película Captain America: The First Avenger?'
respuesta_3 = clasificar_pregunta(pregunta_3, clave_huggingface=clave_huggingface)
print(respuesta_3)

<class 'str'>
Película: [Captain America: The First Avenger]
```

17. Resultados de clasificar cada tipo de pregunta

Al realizar preguntas relacionadas con la historia o definiciones en el ámbito de la literatura de superhéroes, se observa que el modelo demuestra capacidad para clasificar adecuadamente estas consultas, aunque brinda información de más, aún cuando le pedí al modelo que no lo haga. A pesar de que en este ejemplo solo incluí una pregunta concreta, he realizado varias pruebas similares dentro de la misma categoría, y el modelo ha logrado clasificar correctamente la gran mayoría de ellas.

Luego, llevé a cabo una evaluación similar para preguntas específicas sobre superhéroes y sus adaptaciones cinematográficas, y el modelo demostró nuevamente capacidad para realizar clasificaciones precisas.

Cabe aclarar que, durante las pruebas, confirmé que la solicitud al modelo de incluir el nombre del superhéroe y la película entre corchetes me permitió obtener el mismo en un formato que más adelante me facilitó la búsqueda en Wikidata y en el Dataframe. A pesar de que exploré otras opciones como %, & y \$, la inclusión de corchetes resultó ser la única forma efectiva de garantizar la obtención del formato necesario para evitar complicaciones futuras.

Ahora que el modelo es capaz de clasificar correctamente las preguntas en sólo una o dos palabras, he desarrollado la función '**obtener_contexto**', diseñada para tomar decisiones basadas en dichas clasificaciones. (Imagen 18)

Cuando la pregunta se refiere a temas de historia o definiciones en el ámbito de la literatura de superhéroes, la función busca en la base de datos vectorial los cinco fragmentos ('chunks') con mayor similitud y los devuelve como contexto en formato de cadena.

En el caso de preguntas que incluyen un superhéroe específico (identificado entre corchetes, de allí la elección de formato para facilitar la búsqueda), la función extrae el nombre del superhéroe y llama a la función '**contexto_grafos**', que consulta la base de datos de grafos online y devuelve información relevante sobre ese superhéroe como contexto.

Finalmente, cuando la pregunta se centra en adaptaciones cinematográficas de superhéroes (películas), la función invoca a '**contexto_pelicula**', que genera un contexto en formato de cadena conteniendo información sobre la película mencionada.


```
def obtener_contexto(respuesta, pregunta_usuario = None):
    # Divido la respuesta en palabras
    palabras = respuesta.split()

    # Si la pregunta es sobre historia/definiciones --> BBDD ChromaDB
    if palabras and palabras[0] == "Historia:":
        contexto = ''
        chroma_client = chromadb.Client()
        collection = chroma_client.get_collection(name="superheroes")
        query_embedding = modelo_embeddings.encode(pregunta_usuario).tolist() # vectorizo pregunta del usuario
        results = collection.query(query_embeddings= [query_embedding], n_results=5) # obtengo primeros 5 chunks
        for result in results['documents'][0]:
            contexto += result

        return (contexto, "Base de datos vectorial")

    # Si la pregunta es sobre un superhéroe específico --> BBDD de Grafos
    elif palabras and palabras[0] == "Superhéroe:":
        # Busco el nombre del superhéroe entre corchetes
        indice_corchete_inicio = respuesta.find("[")
        indice_corchete_fin = respuesta.find("]", indice_corchete_inicio)

        if indice_corchete_inicio != -1 and indice_corchete_fin != -1:
            nombre_superheroe = respuesta[indice_corchete_inicio + 1:indice_corchete_fin]
            contexto_personaje = contexto_grafos(nombre_superheroe)
            return (contexto_personaje, 'Base de datos de grafos')

        else:
            return "Superhéroe", None

    # Si la pregunta es sobre una película --> Datos Tabulares (Dataframe)
    elif palabras and palabras[0] == "Película:":
        # Busco el nombre de la película entre corchetes
        indice_corchete_inicio = respuesta.find("[")
        indice_corchete_fin = respuesta.find("]", indice_corchete_inicio)

        if indice_corchete_inicio != -1 and indice_corchete_fin != -1:
            nombre_pelicula = respuesta[indice_corchete_inicio + 1:indice_corchete_fin]
            contexto_pelicula_elegida = contexto_pelicula(nombre_pelicula)
            return (contexto_pelicula_elegida, 'Datos Tabulares')

        # Si la respuesta no tiene el formato esperado
        else:
            return "Categoría no identificada", None
```

18. Función 'obtener_contexto', la cual obtiene el contexto según la clasificación de la pregunta

A continuación, se muestra que para cada tipo de pregunta, se obtiene la fuente de datos esperada. (Imagen 19)

```
# Pregunta sobre historia/definiciones (BBDD vectorial)

contexto_1, fuente_1 = obtener_contexto(respuesta_1, pregunta_1)

print("Contexto: ", contexto_1, "\n")
print("Fuente elegida: ", fuente_1)

Contexto:  hecho 1961 editorialque hacia tiempo publicaba historietas superher
Fuente elegida: Base de datos vectorial
```

```
# Pregunta sobre un superhéroe específico (BBDD de Grafos)

contexto_2, fuente_2 = obtener_contexto(respuesta_2, pregunta_2)

print("Contexto: ", contexto_2, "\n")
print("Fuente elegida: ", fuente_2)

Contexto: Spider-Man :
sex or gender: male
country of citizenship: United States of America
birth name: Peter Benjamin Parker
place of birth: Queens
father: Richard Parker

# Pregunta sobre una película (Datos Tabulares)

contexto_3, fuente_3 = obtener_contexto(respuesta_3, pregunta_3)

print("Contexto: ", contexto_3, "\n")
print("Fuente elegida: ", fuente_3)

Contexto: Película: Captain America: The First Avenger
Fecha de lanzamiento: 22/07/2011
Ingresos fin de semana apertura US: 65058524
Ingresos US: 176654505
Ingresos mundiales: 370569774
Duración: 124 minutos
```

19. Ejemplos de obtención de contexto según el tipo de pregunta

Finalmente, definí la función '**chatbot**', la cual recibe un prompt y un contexto. (Imagen 20)

Dicha función simula una interacción entre un usuario y un asistente. Se utiliza la plantilla `zephyr_chat_template` para estructurar la conversación. La información de contexto se proporciona al asistente, y luego se plantea una pregunta específica al modelo (la ingresada por el usuario). La función realiza una solicitud a la API de Hugging Face para generar una respuesta coherente y basada en hechos en función del contexto y la pregunta proporcionados.

La URL de la API de Hugging Face, las cabeceras y los datos para la solicitud POST se configuran adecuadamente. Después de realizar la solicitud, se extrae y devuelve la respuesta generada por el modelo, considerando la información de contexto y la pregunta proporcionada.

En caso de que ocurra algún error durante el proceso, se maneja y se imprime un mensaje indicando que se produjo un error.

Nuevamente, el modelo que decidí utilizar es `zephyr-7b-beta` de Hugging Face, el cual es el segundo modelo de la serie Zephyr y está entrenado para actuar como un asistente útil.

```
def chatbot(prompt: str, context: str, clave_huggingface, max_new_tokens: int = 768) -> None:
    '''Recibe una pregunta (prompt) y un contexto y devuelve una respuesta en base a
    ambos'''
    messages: List[Dict[str, str]] = [
        {
            "role": "system",
            "content": "Eres un asistente útil que siempre responde con respuestas veraces, útiles y basadas en hechos."
        },
        {
            "role": "user",
            "content": f"La información de contexto es la siguiente:\n\
            -----\n\
            {context}\n\
            -----\n\
            Dada la información de contexto anterior, y sin utilizar conocimiento previo, responde la siguiente pregunta.\n\
            Pregunta: {prompt}\n\
            Respuesta: "
        }
    ]

    try:
        prompt_formatted: str = zephyr_chat_template(messages, add_generation_prompt=True)

        # URL de la API de Hugging Face para la generación de texto
        api_url = "https://api-inference.huggingface.co/models/HuggingFaceM4/zephyr-7b-beta"

        # Cabeceras para la solicitud
        headers = {"Authorization": f"Bearer {clave_huggingface}"}

        # Datos para enviar en la solicitud POST
        # Sobre los parámetros: https://huggingface.co/docs/transformers/main\_classes/text\_generation
        data = {
            "inputs": prompt_formatted,
            "parameters": {
                "max_new_tokens": max_new_tokens,
                "temperature": 0.7,
                "top_k": 50,
                "top_p": 0.95
            }
        }

        # Realizo la solicitud POST
        response = requests.post(api_url, headers=headers, json=data)

        # Extraigo respuesta
        respuesta = response.json()[0]["generated_text"][len(prompt_formatted):]
        return respuesta

    except Exception as e:
        print(f"An error occurred: {e}")
```

20. Función 'chatbot' la cual devuelve una respuesta basada en un contexto y una pregunta

En la siguiente imagen se muestra, para las mismas preguntas de antes, la respuesta del modelo al ser llamado por la función. (Imagen 21)

```
print(f'Pregunta: {pregunta_1}\n')
print(f'Respuesta:\n {chatbot(pregunta_1, contexto_1, clave_huggingface=clave_huggingface)}')

Pregunta: ¿Cuándo fue el inicio de los cómics?

Respuesta:
La pregunta se refiere a los cómics como formato de historietas, en lugar de la industria de cómics como un todo. Según la informac

print(f'Pregunta: {pregunta_2}\n')
print(f'Respuesta:\n {chatbot(pregunta_2, contexto_2, clave_huggingface=clave_huggingface)}')

Pregunta: ¿Cuál es el nombre completo de Spider-Man?

Respuesta:
El nombre completo de Spider-Man según la información de contexto proporcionada es Peter Benjamin Parker.

print(f'Pregunta: {pregunta_3}\n')
print(f'Respuesta:\n {chatbot(pregunta_3, contexto_3, clave_huggingface=clave_huggingface)}')

Pregunta: ¿Cuánto dura la película Captain America: The First Avenger?

Respuesta:
La duración de la película Captain America: The First Avenger es de 124 minutos, según la información de contexto proporcionada.
```

21. Respuestas brindadas por el modelo al ser llamada la función 'chatbot'

Finalmente, creé una función llamada '**hacer_pregunta_interactiva**', que permite interactuar con el chatbot haciéndole alguna pregunta de interés acerca de superhéroes. (Imagen 22)

```
def hacer_pregunta_interactiva():
    while True:
        # Solicito al usuario que ingrese una pregunta
        print("Hola, soy un chatbot especializado en superhéroes. ¡Hazme una pregunta! \n")
        pregunta_usuario = input("Ingresa tu pregunta o escribe 'salir' para salir: ")

        # Verifico si el usuario desea salir
        if pregunta_usuario.lower() == 'salir':
            print("¡Hasta luego!")
            break

        # Clasifico la pregunta
        respuesta = clasificar_pregunta(pregunta_usuario, clave_huggingface=clave_huggingface)

        # Obtengo contexto según el tipo de pregunta
        contexto_usuario, fuente_usuario = obtener_contexto(respuesta, pregunta_usuario)

        # Llamo a la función chatbot con la pregunta y el contexto proporcionados
        respuesta_chatbot = chatbot(pregunta_usuario, contexto_usuario, clave_huggingface=clave_huggingface)

        # Muestro la pregunta y la respuesta generada por el chatbot
        print(f'\nPregunta: {pregunta_usuario}\n')
        print(f'Respuesta:\n {respuesta_chatbot}\n')

    # Llamo a la función para comenzar la interacción
    hacer_pregunta_interactiva()
```

22. Función interactiva para hacerle preguntas al chatbot

A continuación, se muestra un ejemplo de la ejecución de la función interactiva. (Imagen 23)

```
Hola, soy un chatbot especializado en superhéroes. ¡Hazme una pregunta!

Ingresar tu pregunta o escribe 'salir' para salir: ¿Cómo es la vestimenta de los superhéroes?
<class 'str'>

Pregunta: ¿Cómo es la vestimenta de los superhéroes?

Respuesta:
La vestimenta de los superhéroes es variada y refleja sus poderes, capacidades y personalidad. En algunos casos, se inspiran en trajes de alta costura o ropa deportiva de alto rendimiento, mien

Hola, soy un chatbot especializado en superhéroes. ¡Hazme una pregunta!

Ingresar tu pregunta o escribe 'salir' para salir: ¿Cuál es el nombre completo de Spider-Man?
<class 'str'>

Pregunta: ¿Cuál es el nombre completo de Spider-Man?

Respuesta:
El nombre completo de Spider-Man según la información de contexto proporcionada es Peter Benjamin Parker. Spider-Man es solo un alias y se refiere a la identidad superheroica de Peter Parker.

Hola, soy un chatbot especializado en superhéroes. ¡Hazme una pregunta!

Ingresar tu pregunta o escribe 'salir' para salir: ¿Cuál fue la fecha de lanzamiento de la película Captain America: Civil War?
<class 'str'>

Pregunta: ¿Cuál fue la fecha de lanzamiento de la película Captain America: Civil War?

Respuesta:
La fecha de lanzamiento de la película Captain America: Civil War fue el 6 de mayo de 2016, según la información de contexto proporcionada.

Hola, soy un chatbot especializado en superhéroes. ¡Hazme una pregunta!

Ingresar tu pregunta o escribe 'salir' para salir: salir
¡Hasta luego!
```

23. Ejemplo del funcionamiento de la función interactiva

Aquí finaliza la resolución del ejercicio 1.

3. EJERCICIO 2

- **Agentes inteligentes**

Si bien existen muchas definiciones para el concepto ‘agente inteligente’, considero que la mejor forma de definirlo es como aquél sistema que es capaz de interpretar y procesar la información proveniente de su entorno, generalmente con el objetivo de satisfacer las necesidades de un usuario o programa.

Sus características fundamentales incluyen la capacidad de aprender de forma autónoma basándose en la información que recibe, interactuar de manera dinámica con su entorno, colaborar eficientemente con otros agentes, tomar decisiones autónomas y ajustar su comportamiento según la información percibida del entorno. En esencia, un agente inteligente opera de manera determinada, pero su adaptabilidad y habilidad para aprender lo distinguen como un componente esencial en entornos dinámicos y complejos.

En la actualidad, la utilidad de los agentes inteligentes es prácticamente ilimitada. En el sector empresarial, las interfaces de conversación impulsadas por inteligencia artificial desempeñan un papel crucial. Los usuarios interactúan cada vez más con estas interfaces para realizar tareas con rapidez, lo que convierte a las aplicaciones de inteligencia artificial conversacionales en un diferenciador competitivo para muchas empresas.

Dentro del mismo rubro, destacan casos de uso específicos, como el servicio al cliente y la optimización de centros de llamadas para una comunicación más efectiva con los clientes. Además, se observa el empleo de agentes inteligentes en diversas áreas, como los asistentes de voz para automóviles y aquellos encargados del mantenimiento preventivo y correctivo de maquinaria de diferentes tipos.

Por otro lado, también son utilizados en ámbitos como el transporte, para optimizar el flujo de tráfico y proporcionar ayuda a la navegación; la educación, al proporcionar experiencias de aprendizaje personalizadas y evaluar el rendimiento de los estudiantes; y en salud, para ofrecer recomendaciones de tratamiento personalizadas y colaborar en los diagnósticos médicos.

Esta versatilidad demuestra la capacidad de los agentes inteligentes para adaptarse a diversas necesidades y desafíos, consolidándose como herramientas fundamentales en la mejora de la eficiencia y la calidad en distintos contextos.

Existen varios tipos de agentes inteligentes, cada uno con características y funcionalidades específicas. Aquí mencionaré algunos de ellos:

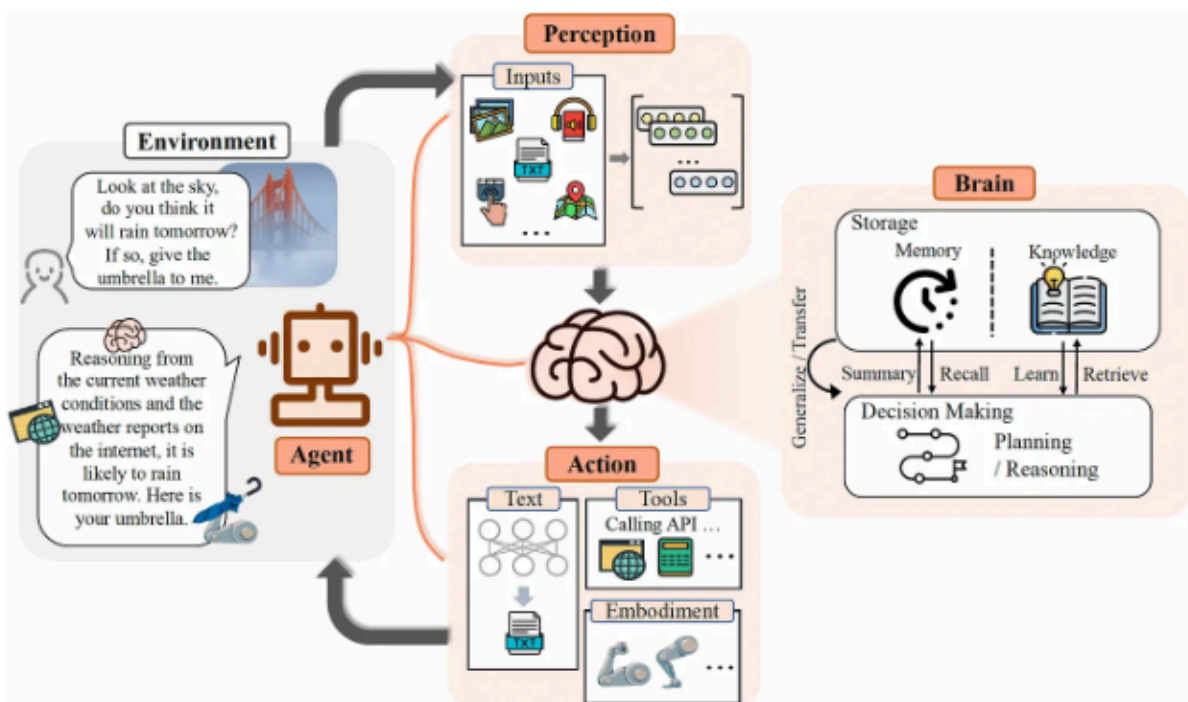
- **Agentes Reactivos Simples:** Son la forma más básica de agentes inteligentes. Actúan en respuesta a eventos específicos en su entorno, siguiendo patrones predefinidos. Son eficientes para tareas específicas y se utilizan comúnmente en sistemas de respuesta automática, como los

asistentes de correo electrónico que clasifican mensajes en categorías predefinidas.

- **Agentes Reactivos Basados en Modelos:** A diferencia de los anteriores, estos incorporan representaciones internas del entorno. Estos modelos les permiten anticipar eventos futuros y tomar decisiones en consecuencia. Por ejemplo, los sistemas de conducción autónoma utilizan modelos para prever el movimiento de otros vehículos y tomar decisiones en tiempo real.
- **Agentes Basados en Objetivos:** Estos agentes tienen metas específicas y toman decisiones orientadas a lograr esos objetivos. Son comunes en entornos dinámicos donde es crucial adaptarse a cambios en el entorno. Un ejemplo práctico son los sistemas de gestión de inventario que buscan maximizar la eficiencia logística para alcanzar objetivos específicos.
- **Agentes que Aprenden:** Se caracterizan por su capacidad de aprendizaje. Mejoran su rendimiento a lo largo del tiempo mediante la experiencia y la retroalimentación. Ejemplos: asistentes personales que comprenden el lenguaje natural y mejoran su capacidad de respuesta con el tiempo.

Cada tipo de agente tiene sus propias ventajas y se adapta mejor a ciertos contextos y aplicaciones, contribuyendo así a la diversidad y eficacia de los sistemas de inteligencia artificial.

A continuación, se presenta una representación visual que ilustra la interacción de un agente inteligente basado en LLM con su entorno. Este agente se involucra en un proceso dinámico de percepción, donde capta información del ambiente, y posteriormente, toma acciones fundamentadas en la información adquirida.



Los LLM actúan como el núcleo cerebral dentro del sistema del agente, descomponiendo tareas complejas en tareas manejables, estrategizando y ejecutando iterativamente cada tarea hasta llegar a una solución. Su capacidad de razonamiento es fundamental para interpretar el contexto cuando acceden a información contextual desde la memoria y recursos externos.

Dentro del agente autónomo, los LLM asumen roles clave como planificación, razonamiento, ejecución, evaluación y resumen. Aprovechan su capacidad de razonamiento lingüístico, sentido común y comprensión para desplegar herramientas que potencian su desempeño en diversas actividades. Estos modelos no solo gestionan la complejidad de las tareas, sino que también demuestran una capacidad integral para interpretar, analizar y abordar problemas en tiempo real.

- **Sistemas multiagentes.**

Un sistema multiagente se compone de varios agentes que interactúan entre sí para lograr un objetivo común. Este enfoque se aplica en diversos campos, como inteligencia artificial, economía y sociología.

Los sistemas multiagentes presentan ventajas significativas en comparación con agentes individuales. En primer lugar, son altamente escalables y capaces de abordar tareas más complejas. En segundo lugar, demuestran una mayor robustez al poder tolerar el fallo de un agente individual. Por último, aprovechan la diversidad de agentes para abordar problemas de manera más eficiente.

No obstante, estos sistemas también enfrentan desafíos importantes. En primer lugar, el diseño y la gestión de un sistema multiagente pueden ser complejos. En segundo lugar, los agentes pueden tener objetivos conflictivos, lo que requiere coordinación. Además, los agentes deben ser capaces de aprender y adaptarse a condiciones cambiantes.

A pesar de estos desafíos, los sistemas multiagentes representan un enfoque poderoso para abordar problemas complejos.

- **Aplicaciones de agentes basados en LLM de código abierto.**

En la actualidad, existen diversas aplicaciones que permiten la creación de agentes inteligentes de código abierto basados en LLM, y Autogen se destaca como ejemplo. Este framework de código abierto utiliza modelos de lenguaje de gran escala para desarrollar múltiples agentes que colaboran para lograr objetivos específicos.

Autogen se orienta a simplificar y facilitar la comunicación entre los agentes, con el objetivo de reducir errores y maximizar el rendimiento de los LLM. Además, brinda características personalizables que permiten a los usuarios seleccionar sus modelos preferidos, mejorar los resultados mediante retroalimentación humana y agregar herramientas adicionales según las necesidades específicas.

ShortGPT es otro ejemplo destacado de un framework de código abierto que aprovecha LLM para abordar tareas complejas. Este framework es especialmente eficaz en la resolución de desafíos vinculados a la creación de videos, síntesis del habla y edición.

Entre las capacidades destacadas de ShortGPT se incluyen la capacidad para gestionar diversas tareas relacionadas con videos, tales como la redacción de guiones para videos, la generación de voiceovers, la selección de música de fondo, la creación de títulos y descripciones, e incluso la edición de videos en sí. ShortGPT se adapta tanto a videos de duración extensa como a aquellos más breves, ofreciendo una versatilidad significativa en la producción de contenido audiovisual.

Otros ejemplos de este tipo de aplicaciones incluyen: ChatDev, SuperAGI, JARVISetc.

Problemática a solucionar.

Como la mayor utilidad que destaque de los sistemas multiagentes es solucionar una tarea que resulta repetitiva, la problemática que planteé solucionar es la siguiente:

Supongamos que usted es el propietario de un negocio o empresa y gestiona toda la información relacionada con las ventas en una hoja de cálculo o base de datos, estableciendo objetivos, por ejemplo, de forma quincenal. Con el fin de evitar la labor repetitiva de generar manualmente un informe que resuma la situación actual de su negocio, se plantea la posibilidad de contar con un sistema multiagente que se encargue de realizar esta tarea de manera automatizada.

En resumen, mi propuesta consiste en implementar un sistema multiagente diseñado para generar informes descriptivos sobre las ventas del negocio, tomando como base los objetivos establecidos.

Los agentes que podrían intervenir son:

- Agente Recolector de Datos (Agente 1): Este agente se encarga de extraer toda la información relevante almacenada en su base de datos, como registros de ventas en formatos como Excel o bases de datos relacionales.
- Agente Redactor de Informes (Agente 2): Una vez que el Agente Recolector ha recopilado los datos, este agente recibe la información y elabora un informe descriptivo. El informe puede incluir detalles como las ventas quincenales en pesos y dólares (obtenidos mediante el Agente 3), comparativas con el periodo anterior, y otros indicadores relevantes. Además, este agente tiene la capacidad de evaluar si se cumplió el objetivo quincenal de ventas y reflejar esta información en el informe.

- Agente Conversor de Divisas (Agente 3): Este agente consulta la página del Banco Nación para obtener el precio de compra del dólar y realiza la conversión correspondiente, proporcionando al Agente Redactor la información necesaria en dólares.

Este esquema puede ampliarse y complejizarse según las necesidades específicas del negocio. Por ejemplo, podrían incorporarse agentes adicionales especializados en áreas como producción, marketing (para informes sobre gastos publicitarios), logística, entre otras. Esta estructura multiagente ofrece la flexibilidad necesaria para adaptarse a diferentes contextos y optimizar la toma de decisiones empresariales mediante la automatización inteligente de tareas.

Para este caso particular, el usuario no interviene en la conversación, sino que es iniciada por el agente 1, cada quince días.

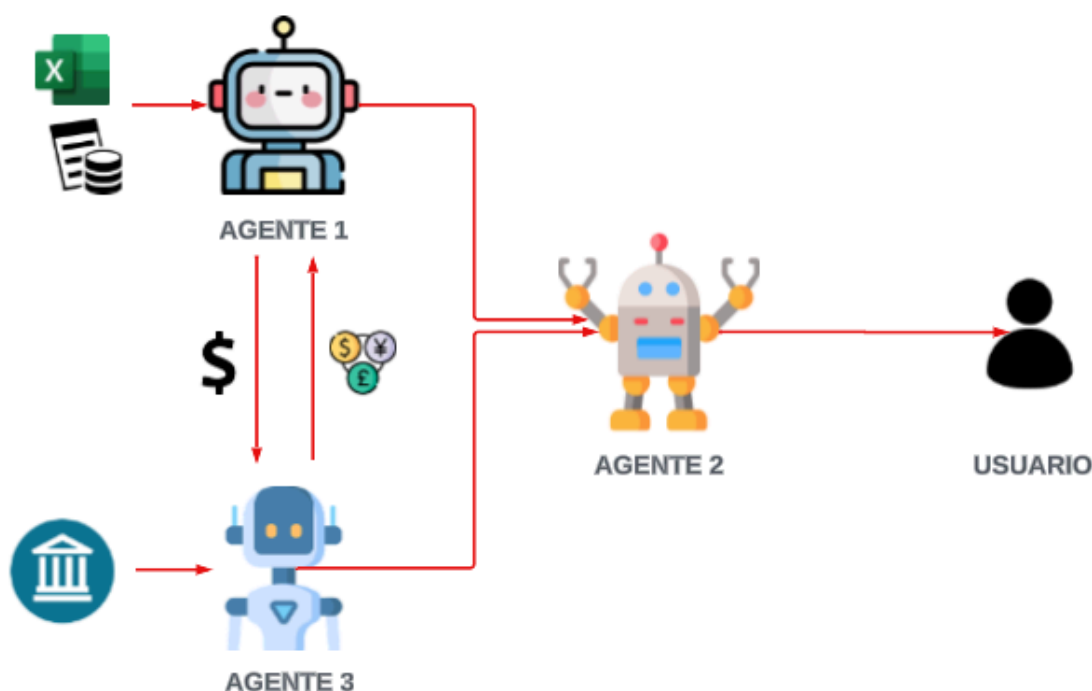
El siguiente es un ejemplo de conversión entre los agentes:

Agente 1: (Hacia el Agente 2) Aquí tienes los datos de las ventas de esta quincena.
(Hacia el Agente 3) Este fue el total de ventas registrado durante este periodo.

Agente 3: // Consulta el valor actual del dólar y realiza la conversión // (Hacia el Agente 2) Aquí está el valor total de ventas expresado en dólares.

Agente 2: Recibido. Procederé a generar el informe.

Finalmente, presento un diagrama descriptivo que detalla el funcionamiento del sistema.



Aquí finaliza la resolución del ejercicio 2.

3.1. FUENTES.

- Fowler, G. A. (2023, October 19). *Revolutionizing AI: The Era of Multi-Agent Large Language Models*. Medium; Medium.
<https://gafowler.medium.com/revolutionizing-ai-the-era-of-multi-agent-large-language-models-f70d497f3472>
- *What is multi-agent system (MAS)? | Autoblocks Glossary*. (2024). Autoblocks.ai. <https://www.autoblocks.ai/glossary/multi-agent-system>
- Blog de CEUPE. (2022, February 10). *¿Qué es un Agente Inteligente? Características, tipos y cómo funciona*. Ceupe; Ceupe.
<https://www.ceupe.com/blog/agente-inteligente.html>
- *AGENTES INTELIGENTES*. (2017, March 14). INTELIGENCIA ARTIFICIAL; INTELIGENCIA ARTIFICIAL.
<https://sitiointeligenciaa.wordpress.com/agentes/>
- Wang, Y. (2023, October 9). *A Complete Guide to LLMs-based Autonomous Agents (Part I)*: Medium; Medium.
<https://medium.com/@yulemoon/a-complete-guide-to-llms-based-autonomous-agents-part-i-69515c016792>