TRABAJO PRÁCTICO FINAL

Ulises Nemeth

Procesamiento de Lenguaje Natural.

Profesores:

Juan Pablo Manson

Alan Geary

Constantino Ferrucci

Dolores Sollberger

Fecha de primer entrega: 18/12/24

Fecha de segunda entrega: 29/12/24

Benchmark de los clasificadores y desarrollo del agente ReAct.

Fecha de tercer entrega: 31/12/24

Clasificador BERT, optimizacion del reranker con uso de palabras clave, mas ejemplos de interacciones con el agente y RAG (Al usar LLMs no deterministicos, las respuestas difieren del apartado del colab).

Indice

1. Resumen General

En este trabajo se presenta la tarea (y los conflictos) de aplicar los conocimientos adquiridos al intentar desarrollar un chatbot y un Agente basados en la técnica **RAG (Retrieval Augmented Generation)** para el juego de mesa "The White Castle". Hay tres fuentes principales de datos: documentos de texto, datos tabulares y bases de datos de grafos. A través de una combinación de herramientas y modelos de PLN, se implementó un clasificador que permite determinar la fuente adecuada para generar respuestas precisas. Se compararon dos versiones del clasificador (basado en LLM y embeddings), optimizando la eficiencia y relevancia en la recuperación de información. Además, se diseñó un agente ReAct para combinar las capacidades del sistema. Al final, los resultados obtenidos, una interpretación acerca de los logros en contraste con la consigna y las limitaciones identificadas.

2. Ejercicio 1

2.1 Resumen

En este apartado se va a desarrollar el webscrapping en dos partes ya que el contenido se extrajo de páginas web y videos. Luego, en tres partes, la utilización de los archivos creados para crear y llenar las fuentes de datos y las problemáticas que surgieron. Al final los clasificadores, retrievers y ReRank y una demostración del chatbot utilizando todos los recursos.

2.2 Introducción

Este primer ejercicio abarca casi la totalidad del trabajo, esto se va a ver reflejado en este informe. Si bien mientras el trabajo estaba en desarrollo no seguía un orden, mientras lo estaba terminando acomodé las cosas para que esta parte sea más ordenada.

Contexto y justificación: Los chatbots con RAG mejoran la generación de respuestas al integrar diversas fuentes de datos. Se busca crear un chatbot que responda preguntas sobre reglas, estrategias y estadísticas del juego The White Castle.

Objetivos específicos:

1. Implementar RAG con múltiples fuentes de datos.
2. Desarrollar dos clasificadores y comparar sus resultados.
3. Optimizar la búsqueda de información mediante queries dinámicas.

Estructura del informe: Se describen los métodos utilizados, el proceso de implementación y los resultados obtenidos.

Metodologia

Selección y Análisis de Fuentes

- **Documentos Textuales:** Reseñas y tutoriales desde youtube y páginas web (ver referencias)
- **Datos Tabulares:** Estadísticas de bgg.
- **Base de Datos de Grafos:** Relación entre diseñadores, mecánicas y artistas modelada con **Neo4j**.

Procesamiento de Datos

- **Segmentación de Textos:** Uso de **LangChain** para dividir textos en chunks.
- **Limpieza de Datos:** Eliminación de redundancias y normalización de texto.
- **Embeddings:** Generación de vectores all-MiniLM-L6-v2.
- **Datos Tabulares y Grafos:** Consultas dinámicas utilizando SQL y Cypher.

Herramientas Utilizadas

- **LangChain** para dividir textos.
- **ChromaDB** para almacenamiento vectorial.
- **Neo4j** para grafos.
- **Pandas/SQLite** para tablas.

Pasos seguidos:

- WebScrapping
- Fuentes de datos
- Clasificadores
- Queries dinamicas
- Busqueda hibrida
- Creación del ChatBot

Desarrollo/Implementación

Los pasos fueron desarrollados en orden secuencial como se describen a continuación:

WebScrapping:

La obtención de texto y datos de forma automatizada aprovechando los recursos publicados online es una forma poco costosa de acceder a grandes volúmenes de información.

En este trabajo se ve un ejemplo de cómo en segundos se pueden obtener los subtítulos de algún video de youtube que tenga una gran profundidad sobre un tema específico. Gracias al PLN se puede conseguir procesar estos datos y darles un propósito sin necesidad de consumirlo por completo de forma tradicional.

También en el desarrollo se consigue el texto de párrafos en páginas web gracias al HTML, mirando el código fuente de sitios relacionados al juego The White Castle, se pueden obtener todo el contenido necesario omitiendo partes no tan relevantes.

Así como se consiguió texto dentro de la estructura html también se extrajo datos de tipo clave valor, con estadísticas del juego para llenar una fuente de datos tabular SQLite.

Luego también, se extrajo contenido con sus respectivas relaciones, del mismo modo que para las estadísticas, pero esta vez para información acerca de los creadores y más datos acerca del desarrollo del juego. Al tener estos datos va a ser posible generar triadas para posteriormente insertarlas en la base de datos de grafos

Fuentes de datos:

Están categorizadas por su contenido, de esta forma se pueden filtrar mejor los datos para dar una respuesta precisa a la hora de interactuar con el usuario.

- **Documentos de texto:** Reglas del juego, reseñas y tutoriales.
- **Tablas:** Estadísticas del juego y datos relevantes (SQLite).
- **Grafos:** Relaciones entre diseñadores y elementos del juego.

Algunos de los documentos de texto ya estaban divididos por párrafos dentro de la estructura html, por esto no fue necesario dividirlos en chunks, ya que tenían su propio contexto ya formado. Otros eran subtítulos extraídos de videos de youtube, los cuales fueron unidos descartando la información de tiempo de aparición y duración de los archivos srt. Para estos archivos de texto si se realizó una división en chunks.

Una vez generado el listado de chunks con sus respectivas fuentes, se hicieron los embeddings utilizando el modelo `all-MiniLM-L6-v2` ya que es uno de los más chicos y que menos recursos utiliza, muy eficiente en el uso de tiempo y memoria. Los embeddings se insertaron en una colección de chromadb. Luego, con una función `query_chroma` se hace una consulta

para obtener chunks especificando la pregunta del usuario y la fuente de datos a utilizar como contexto (obtenida por el clasificador).

Para la creación de las tablas en SQLite, se utilizaron los datos estructurados con contenido estadístico. Se agruparon los contenidos similares en tablas con dos columnas “key” y “value”. Más adelante se va a explicar la utilización de estos datos.

Lo mismo para la creación de grafos, los datos obtenidos de forma estructurada se segmentaron en formatos clave valor, donde todos estos estaban relacionados con el juego. Utilizando la descripción como relación y los valores como entidad.

The White Castle	姬路城	Alternate Names
The White Castle	白鷺城	Alternate Names
The White Castle	백로성	Alternate Names
The White Castle	2023	Year Released
The White Castle	Isra C.	Designers
The White Castle	Shei S.	Designers

Ejemplo de triadas.

<u>Prueba de LLMs:</u>

Luego de probar varios LLM de huggingface remotamente, decidí quedarme con Qwen/Qwen2.5-Coder-32B-Instruct para varias de las tareas en las que necesite un LLM. Los motivos fueron:

1. Precisión en las respuestas:

Comparado con otros modelos probados, Qwen fue más preciso y coherente en las respuestas, incluso para preguntas abiertas y contextos variados. Por ejemplo: En la pregunta sobre GPUs con más CUDA Cores, Qwen identificó correctamente la Nvidia GeForce RTX 4090 como una GPU de alta gama actual y nombró específicamente la cantidad, mientras que otros modelos tuvieron que recurrir a contexto enviado por mi o respondian con información obsoleta.

2. Generación natural del lenguaje:

El modelo genera respuestas bien estructuradas y comprensibles. En la pregunta "¿Cómo se hace un omelette?", Qwen no solo proporciona instrucciones claras y organizadas, sino que también incluyó detalles útiles como ingredientes y técnicas de cocción. Esto contrasta con respuestas menos claras o incluso ininteligibles de otros modelos (como Falcon-7B).

3. Adaptabilidad y profundidad:

Qwen tiene la capacidad de adaptarse a distintos temas con mayor profundidad. Por ejemplo, en la pregunta sobre filamentos 3D, su respuesta incluyó una variedad de opciones (filamentos metálicos, de carbono, biomédicos, etc.), demostrando que tiene un contexto más amplio y técnico del tema.

4. Uso en tareas prácticas:

El modelo es particularmente adecuado para aplicaciones prácticas y contextos donde se requiere precisión técnica, como programación, hardware o procesos paso a paso. Esto lo convierte en una excelente opción para varias de las necesidades de este trabajo.

5. Desempeño general en comparación con otros modelos:

Falcon-7B: Aunque eficiente, tuvo problemas significativos con respuestas inexactas y texto incoherente.

RoBERTa-base-SQuAD2: Requiere contexto adicional, y las respuestas son fragmentadas, con puntuaciones de confianza bajas, limitando su aplicabilidad en entornos reales.

Clasificadores:

Para categorizar la pregunta del usuario, se tienen en cuenta las fuentes de datos extraídas con webscrapping. Las categorías son “Tutorial”, “Reseñas”, “Créditos” y “Estadísticas”. Se pusieron a prueba dos modelos que reciben la query y devuelven la categoría a la que pertenecen. El primero, utilizando el modelo de huggingface `Qwen2.5-Coder-32B-Instruct` recibe un prompt que le ordena elegir entre una de las 4 categorías:

```
prompt_clasificador = f"""
Sos un clasificador experto en juegos de mesa. Clasificá las siguientes preguntas según la fuente más relevante:

Fuentes disponibles:
{fuentes_str}

Pregunta:
{query}

Indica solo la fuente más relevante.
"""
```

De esta forma simple, con las categorías bien adaptadas y diferenciadas, es un LLM el que decide con qué categoría clasificar a la pregunta del usuario.
El segundo modelo, una regresión logística, es entrenado con datos etiquetados. Estos se vectorizan y se dividen en datos de entrenamiento y prueba.

Para ambos modelos se utilizan los mismos datos de prueba. El modelo de regresión logística predijo correctamente el 50% de las categorías, mientras que el modelo de LLM tuvo una precisión del 80%

```
Precisión Regresión Logística: 0.5
Reporte de clasificación Regresión Logística:
              precision    recall  f1-score   support

           0       0.20      1.00      0.33         1
           1       1.00      0.50      0.67         2
           2       1.00      0.50      0.67         4
           3       0.50      0.33      0.40         3

    accuracy                           0.50        10
   macro avg       0.68      0.58      0.52        10
weighted avg       0.77      0.50      0.55        10
```

```
valor real Reseñas
valor real Estadisticas
valor real Reseñas
valor real Estadisticas
valor real Reseñas
valor real Estadisticas
valor real Creditos
valor real Creditos
valor real Tutorial
valor real Reseñas
valor predicho Reseñas
valor predicho Estadisticas
valor predicho Reseñas
valor predicho Estadisticas
valor predicho Reseñas
valor predicho Estadisticas
valor predicho Creditos
valor predicho Tutorial
valor predicho Tutorial
valor predicho Tutorial
Precisión Modelo LLM: 0.8
```

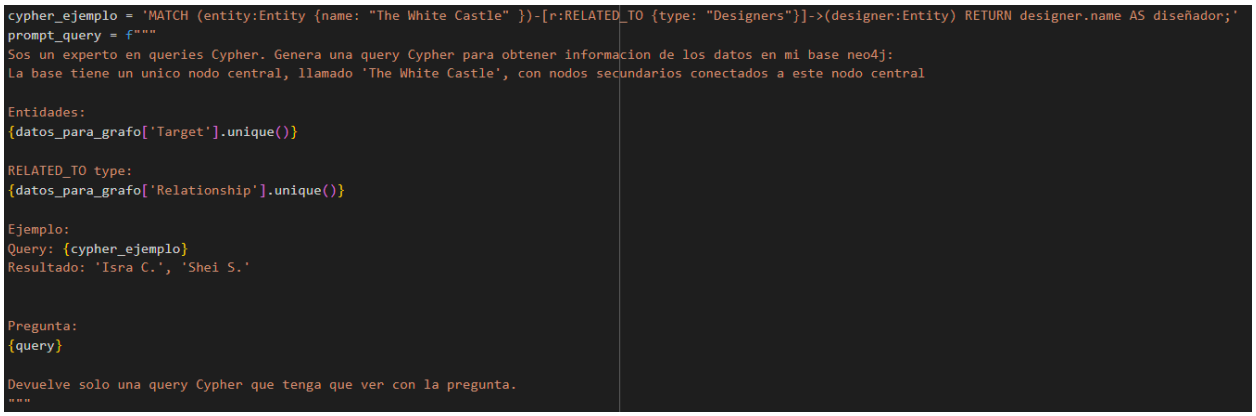
En la tercer entrega se agrego un nuevo clasificador usando BERT (Bidirectional Encoder Representations from Transformers). Captura las relaciones entre palabras, considerando el contexto completo de la oración. Al usar un modelo multilingüe preentrenado, tiene garantizada la compatibilidad con diferentes idiomas. Se puso a prueba con el mismo set de entrenamiento que a los otros clasificadores, pudo predecir correctamente el 70% de los casos.

```
Precisión del modelo: 0.7
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.33      1.00      0.50         1
           1       1.00      1.00      1.00         2
           2       1.00      0.50      0.67         4
           3       0.67      0.67      0.67         3

    accuracy                           0.70        10
   macro avg       0.75      0.79      0.71        10
weighted avg       0.83      0.70      0.72        10
```

Queries dinamicas:

Generar_query_cypher

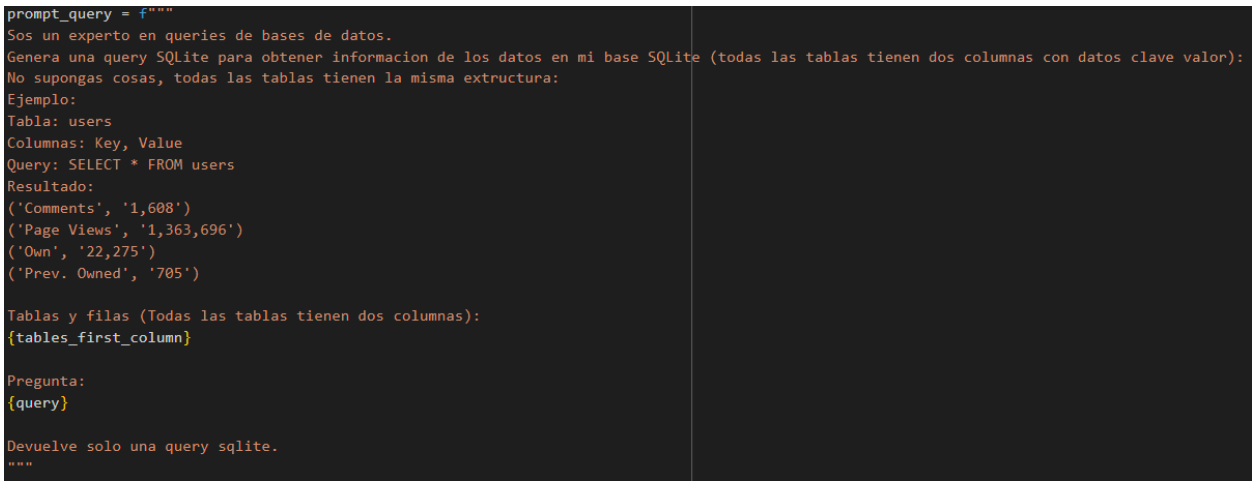
Recibe una pregunta del usuario y genera una query cypher, en el prompt se le da un ejemplo de una query que muestra los diseñadores del juego y se le mencionan las entidades y las relaciones para darle libertad de crear la query para obtener los datos de la base de grafos.



```
cypher_ejemplo = 'MATCH (entity:Entity {name: "The White Castle" })-[r:RELATED_TO {type: "Designers"}]->(designer:Entity) RETURN designer.name AS diseñador;'
prompt_query = f"""
Sos un experto en queries Cypher. Genera una query Cypher para obtener informacion de los datos en mi base neo4j:
La base tiene un unico nodo central, llamado 'The White Castle', con nodos secundarios conectados a este nodo central

Entidades:
{datos_para_grafo['Target'].unique()}

RELATED_TO type:
{datos_para_grafo['Relationship'].unique()}

Ejemplo:
Query: {cypher_ejemplo}
Resultado: 'Isra C.', 'Shei S.'


Pregunta:
{query}

Devuelve solo una query Cypher que tenga que ver con la pregunta.
"""
```

Generar_query_sqlite

Del mismo modo, recibe una pregunta del usuario y genera una query para obtener datos de las tablas de SQLite. Se le menciona cuales son las tablas y los valores de la primera columna de cada tabla.



```
prompt_query = f"""
Sos un experto en queries de bases de datos.
Genera una query SQLite para obtener informacion de los datos en mi base SQLite (todas las tablas tienen dos columnas con datos clave valor):
No supongas cosas, todas las tablas tienen la misma extructura:
Ejemplo:
Tabla: users
Columnas: Key, Value
Query: SELECT * FROM users
Resultado:
('Comments', '1,608')
('Page Views', '1,363,696')
('Own', '22,275')
('Prev. Owned', '705')

Tablas y filas (Todas las tablas tienen dos columnas):
{tables_first_column}

Pregunta:
{query}

Devuelve solo una query sqlite.
"""
```

Ambas queries son creadas por el modelo `Qwen2.5-Coder-32B-Instruct` y también se les aplica una limpieza aprovechando el markup language que usa el modelo para escribir el código en bloques. `respuesta.split("```")` Quedandome con la cadena de caracteres que comienza con “sql” y “cypher” respectivamente, se obtiene un resultado óptimo a la hora de utilizar estas queries.

Busqueda hibrida:

Busqueda semantica:

`Busqueda_semantica` utiliza el modelo `all-MiniLM-L6-v2` para convertir el contexto y la query del usuario en embeddings y utiliza la similitud del coseno para obtener los pasajes más similares.

Búsqueda por palabras clave:

`Busqueda_por_palabras_clave` hace una traducción al inglés de la query del usuario, luego procesa el texto con spaCy y filtra las palabras clave según etiquetas gramaticales y largo de la palabra. Se queda con una instancia de cada palabra para omitir repetidas y las devuelve.

Rerank:

El `reranker` recibe la query del usuario y un listado de pasajes/chunks de texto, los ordena por relevancia y devuelve los 5 más importantes para utilizarse en el contexto de la respuesta del chatbot. (*En la tercer entrega se incorporo la busqueda por palabras clave dentro del reranker.*)

Chatbot:

Se implementan todos los recursos disponibles desarrollados en los pasos anteriores. Se recibe una query del usuario que es clasificada para disparar el camino hacia la fuente de datos correcta. Si se clasifica hacia los tutoriales o reseñas, el reranker acota el contexto que trae la función `query_chroma`, y se lo da al LLM `Qwen2.5-Coder-32B-Instruct` que dado un prompt con el contexto y la pregunta que envió el usuario originalmente, genera una respuesta sin desviarse del objetivo ni utilizando información fuera del contexto proporcionado.

Resultados

1. **Errores comunes:**

El clasificador no funciona siempre correctamente, esto hace que se intenten buscar datos donde no están. Por esto le di la libertad a las queries dinámicas de devolver ‘None’ cuando lo que se quiere averiguar no está presente en las tablas/entidades a buscar. Cuando esto pasa, hago que el modelo intente recurrir al contenido de los archivos de texto como último recurso.

2. **Análisis:**

   Se cumplió con el objetivo del RAG, el chatbot puede recurrir a las fuentes necesarias en cada query para responder con un contexto más acotado de la forma más específica posible.

Ejemplos de Consultas:

Pregunta 1: ¿Quién diseñó The White Castle?

Categoria: Creditos

Respuesta: Basado en el contexto proporcionado, hay dos diseñadores mencionados:

1. Isra C.

2. Shei S.

La pregunta "¿Quién es el diseñador?" es un poco ambigua ya que hay dos diseñadores en la lista. Si necesitas información específica sobre uno de ellos, necesitaría que me indiques cuál es el diseñador que te interesa. ¿Podrías aclarar tu pregunta?

Pregunta 2: ¿Cuántos dados se usan?

Categoría: Datos numéricos (Esta categoría fue cambiada posteriormente a “Estadisticas”)

Respuesta: Basándome en el contexto proporcionado, no tengo información suficiente para determinar cuántos dados se usan. El dato '3.02' no proporciona ningún contexto sobre la cantidad de dados en un juego o en una situación específica. ¿Podrías proporcionar más detalles o contexto?

Pregunta 3: ¿Cuántos jugadores hubo en el último mes?

Categoría: Estadísticas

Respuesta: Basándome en el contexto proporcionado, la respuesta sería que hubo 2,670 jugadores en el último mes. Sin embargo, es importante mencionar que este número se deriva exclusivamente de los datos que me has proporcionado y asumo que este es el dato relevante y actualizado para responder a la pregunta específica. Si necesitas más detalles o tienes más contexto, no dudes en proporcionarlo.

Pregunta 4: ¿Qué es una "acción"?

Categoria: Tutorial

Respuesta: Una "acción" en este juego parece referirse a una tarjeta o una opción que puedes usar para realizar movimientos específicos o para obtener recursos. Aquí te dejo algunos ejemplos de acciones mencionadas en tu contexto:

1. **Colocar una tarjeta en tu tablero.** Esto implica tomar una tarjeta de la habitación donde tu cortesano se encuentra, usarla según las instrucciones, y luego devolverla a su lugar o reemplazarla con otra.

2. **Activar una acción en áreas específicas del tablero.** En ciertas áreas del tablero, como el castillo o las zonas fuera de las murallas del castillo, puedes activar acciones y obtener recursos colocando un dado o una ficha en espacios disponibles.

3. **Realizar "social climbing" (ascenso social).** Con ciertas tarjetas o condiciones, puedes mover uno o dos espacios con tu cortesano o tu cortejo, pagando una cantidad específica de recursos (en este caso, "mother of pearls") para hacerlo.

4. **Realizar acción de "cortejo".** Esto puede implicar colocar un jardinero en el tablero o realizar otras acciones relacionadas con un cortesano, pagando una cantidad de monedas para hacerlo.

En resumen, una acción es una operación específica que puedes realizar en el juego para avanzar tu objetivo, ya sea obteniendo recursos, moviendo tus piezas o activando elementos en el tablero. Cada acción puede requerir diferentes condiciones y costos, y se representa comúnmente mediante el uso de tarjetas o dados

Pregunta 5: ¿Qué problemas tiene el juego según los jugadores?

Categoria: Reseñas

Respuesta: Según los datos proporcionados, los principales problemas que los jugadores suelen enfrentar en The White Castle son principalmente dos:

1. **Ensimismamiento y presión**: El juego requiere una optimización constante de las acciones para maximizar los resultados, lo que puede llevar a los jugadores a quedarse atrapados en la búsqueda de soluciones específicas. Esto a menudo ocurre cuando falta algún elemento crucial y puede llevar a la frustración por la necesidad constante de análisis y automejora ante una alta presión responsable del ritmo acelerado del juego.

2. **Falta de tregua y curva de aprendizaje**: Como se menciona, el juego es intenso y cada turno es importante. Esto puede resultar en experiencias poco satisfactorias en partidas iniciales si los jugadores no están familiarizados con las tácticas adecuadas. Aunque no es que sea intrínsecamente difícil, la complejidad de las jugadas y el ritmo de juego pueden ser desalentadores, especialmente cuando se mezclan jugadores novatos con jugadores experimentados. Esto puede causar frustración entre los jugadores menos experimentados al ver cómo los más hábiles aprovechan mejor las acciones disponibles.

Sin embargo, estos problemas tienden a desvanecerse conforme los jugadores se familiarizan con las mecánicas del juego.

Casos de Error:

Cuando el modelo tiene un contexto sin muchos datos, por ejemplo, con resultados de la primer pregunta, que solamente le llega la query y dos nombres, suele indicar que no está seguro de que responder, ya que el contexto queda muy acotado por la naturaleza de los resultados de fuentes de datos tabulares.

En la segunda pregunta, el clasificador categorizo mal la query del usuario, si bien se está pidiendo un dato numérico, la pregunta hace referencia a la cantidad de dados que se utilizan durante el juego, esta información debería ser buscada en el tutorial. Modificando la categoría de “Datos numéricos” a “Estadísticas”, la query “Cuantos dados se usan?” es categorizada correctamente, ya que el clasificador es un LLM, funciona mejor con esta descripción. Luego busca el contexto en las fuentes de datos correspondientes a los tutoriales.

Conclusiones

1. Resumen de logros:

   Implementación exitosa del chatbot con RAG.

   Desviación a otro material si el principal falla.

2. Evaluación del cumplimiento de objetivos.

   Se consiguió extraer información para alimentar las fuentes de datos.

   Se pudo utilizar un reranker para acotar el contexto a lo más relevante.

   Las queries dinámicas son robustas y devuelven resultados correctamente.

   Se consiguió comparar la precisión de los clasificadores, el clasificador que usa un LLM es superior, probablemente esto cambie si se usan otros datos etiquetados y descripciones de categorías distintas.

3. Recomendaciones:

   Habilitar al usuario a hacer más preguntas para poder indagar sobre el mismo contexto.

   Ampliar las fuentes de datos para enriquecer las respuestas.

3. Ejercicio 2

3.1 Resumen

Se extendió el chatbot a un **agente interactivo** basado en ReAct utilizando **Llama-Index**. El agente combina tres herramientas (`doc_search`, `graph_search`, `table_search`) para responder consultas complejas.

3.2 Introducción

En esta parte del trabajo fue imprescindible el uso del runtime con GPU en la notebook del google colab. Los modelos que se adaptaban al uso del agente ReAct eran pagos a la hora de usarlos remotamente vía API, por ende se necesito correrlos localmente en el entorno. Para el uso correcto de las herramientas se necesita un modelo que sea compatible con ellas.

Objetivo del Trabajo

Desarrollar un agente que razone y actúe utilizando múltiples herramientas dinámicamente.

Justificación

La combinación de herramientas mejora la capacidad del sistema para resolver consultas más complejas.

3.3 Metodología

Se reutilizaron todas las herramientas desarrolladas para el ejercicio 1.

Pasos seguidos:

Desarrollo de Herramientas para el agente:

1. **doc_search:** Busca información en documentos de texto.
2. **graph_search:** Consulta bases de datos de grafos mediante queries Cypher.
3. **table_search:** Genera consultas SQL para recuperar información tabular.
4. **(Intento) traductor:** Traduce textos del español al inglés para enriquecer la comprensión del modelo. *Esta herramienta no está en el notebook, fue un intento para mejorar el comportamiento del agente, ya que repetía constantemente que necesitaba traducir la respuesta al inglés cuando la pregunta se hacía en español. Se quito luego de que el comportamiento sea peor, mientras más herramientas utiliza más probable es que intente crear y utilizar parámetros que no se piden.*

Integración del Agente: Se utilizó el módulo ReActAgent de Llama-Index para combinar las herramientas. Esto permitió que el agente razonara y seleccionara las herramientas más adecuadas para cada consulta. La configuración incluyó un prompt del sistema que prioriza el uso de las herramientas sobre el conocimiento interno del modelo.

Pasos seguidos:

Descarga de dependencias.

Instalación de ollama para correr un modelo compatible con herramientas.

Instancia del LLM.

Creación de herramientas.

Instancia del agente ReAct.

Uso del método chat()

3.4 Desarrollo e Implementación

El desarrollo se llevó a cabo siguiendo los pasos:

Descarga de dependencias:

Cambio del runtime (de CPU a GPU).

Instalación de dependencias necesarias de Llama-Index

Instalación de ollama y LLM:

Se instalo Ollama y se configuro un servidor local para ejecutar el modelo Granite3-dense:2b.

Se probaron varios modelos hasta encontrar uno relativamente chico (2 billones de parámetros) que sea capaz de utilizar herramientas. Qwen lograba responder correctamente pero nunca utilizaba herramientas, llama3.1 no podía utilizarse ya que el runtime del colab no cumplía con los requerimientos mínimos.

Instancia del LLM:

Se instancio el modelo para ser utilizado posteriormente por el agente

Creación de herramientas:

Definición de funciones Python para interactuar con bases de datos, realizar traducciones y consultas contextuales.

Configuración de las herramientas como objetos FunctionTool en Llama-Index.

Instancia del agente ReAct:

Uso de ReActAgent para orquestar las herramientas y gestionar la conversación.

Configuración de verbosidad para el seguimiento del proceso de razonamiento del agente.

Uso del método chat():

Realización de pruebas con consultas simples.

Ajuste de herramientas y prompts para mejorar el comportamiento del agente..

3.5 Resultados

Errores comunes:

Se informó en varias ocasiones que el parámetro de entrada de las herramientas es una cadena de caracteres, a pesar de esto el agente intenta insertar elementos complejos como parámetro de entrada.

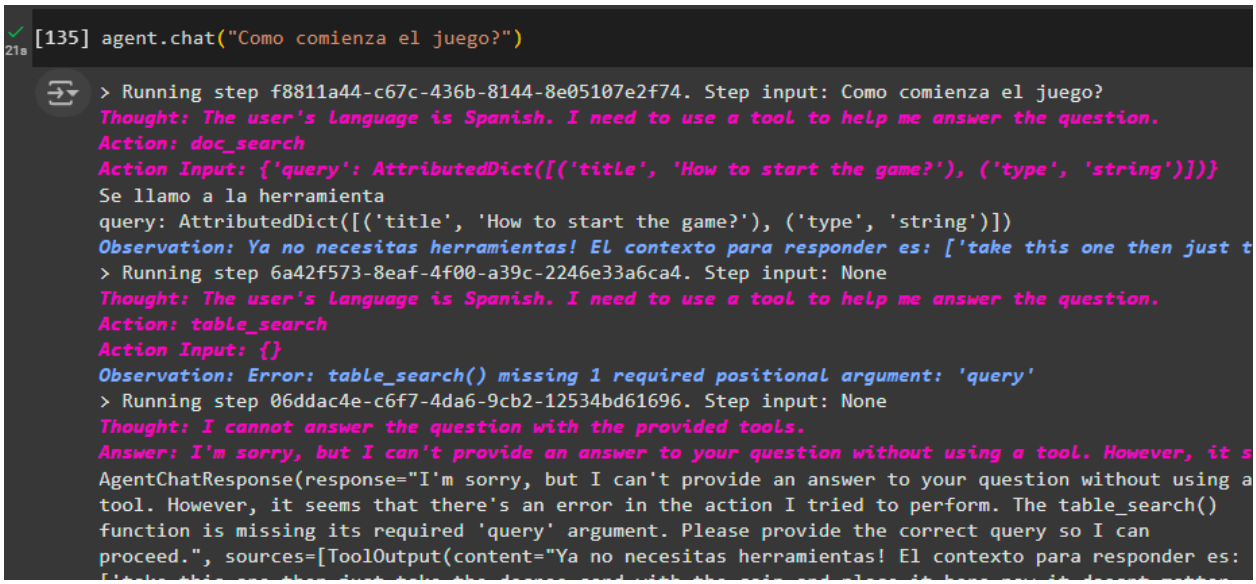
Suele elegir incorrectamente las herramientas a utilizar.

Análisis:

Tiene un comportamiento mas erratico al no tener un listado de herramientas con uso secuencial como el chatbot desarrollado en el ejercicio 1.

En mi caso, la implementación del agente interactivo basado en ReAct demostró ser una solución menos efectiva para responder consultas mediante el uso de herramientas especializadas, seguramente con más tiempo de desarrollo podría alcanzar o mejorar los resultados del RAG.

Consultas:



Aca vemos como el agente usa una herramienta, la cual debería recibir el parámetro de tipo string, pero este termina siendo de tipo AttributedDict. Con las intenciones de que responda con el contexto dado por la herramienta, inserte el texto "Ya no necesitas herramientas!" concatenado al inicio de la respuesta de la herramienta, igualmente intentó usar otra herramienta.

```
Thought: I can answer without using any more tools. I'll use the user's
language to answer
```

```
Answer: The game starts with each player placing their fan token on the
starting space of the scoring track. Players then shuffle the starting
resource cards and action cards, and take turns selecting and taking one
pair of these cards. The turn order is determined by the players'
positions on the turn order track. Each player has a personal board with
their clan members divided into Warriors, gardeners, and courtiers.
Players gain resources by placing their action card in the designated
space and taking the corresponding decree card if it is present. The first
player to reach the final level of their three main resources gains
victory points. The player with the most points at the end of the game
wins.
```

Este es el resultado de hacer la misma pregunta pero en inglés. Luego de usar la herramienta doc_search, es capaz de generar una respuesta.

Pregunta: ¿Quiénes son los diseñadores del juego?

```
Thought: The user is asking about the designers of a game. I need to use a
tool to help me find this information.
Action: doc_search
Action Input: {'query': AttributedDict([('title', 'Designers'), ('type',
'string')])}
Thought: The user is asking about the designers of a game called "The
White Castle". I have used the doc_search tool to find information about
this game. The game was designed by Shea and Ira (also known as the
Yamadays) and has art by Juan Guardiat.
Answer: The White Castle was designed by Shea and Ira (the Yamadays) and
has art by Juan Guardiat.
```

En esta consulta, el agente intentó utilizar las fuentes de datos de tutorial, a pesar de tener otra herramienta con una descripción más atractiva para obtener información acerca de los créditos del juego. Respondió casi correctamente `Shea and Ira` *en lugar de Shea and Isra.*

3.6 Conclusiones

4. Resumen de logros:

   Implementación exitosa del chatbot con RAG.

   Desviación a otro material si el principal falla.

5. Evaluación del cumplimiento de objetivos.
   - Creación de un agente capaz de utilizar herramientas dinámicamente para resolver consultas complejas.
   - Adaptación de las herramientas que ya utilizaba el RAG, a pesar de que el agente intente usarlas de forma incorrecta.
6. Recomendaciones:
   - Agregar más herramientas para perfeccionar el comportamiento.
   - Utilizar modelos con mayor cantidad de parámetros.

4. Referencias.

https://misutmeeple.com/2023/11/resena-the-white-castle/ *Reseña: The White Castle.*

https://pandas.pydata.org/ Pandas.

https://boardgamegeek.com/boardgame/371942/the-white-castle/ *Boardgamegeek.*

https://youtu.be/UDupg6a1GRM *How to play The White Castle - Devir*. Devir Games channel

https://www.langchain.com *LangChain. LangChain Documentation.*

https://chromadb.com *ChromaDB. ChromaDB Documentation.*

https://neo4j.com/ *Neo4j. Graph Database Technology.*

https://huggingface.co/Qwen *Hugging Face. Qwen Models*

https://docs.llamaindex.ai/en/stable/examples/llm/huggingface/ *Hugging Face LLM para agente.*

https://ollama.com/library/granite3-dense/blobs/63dd4fe4571a *Modelo compatible con herramientas Granite3-dense.*

How to play The White Castle boardgame - Full teach + Visuals - Peaky Boardgamer *How to play The White Castle boardgame - Full teach + Visuals - Peaky Boardgamer. Peaky Boardgamer channel*

5. Anexos

1. Descripción del Eurogame: *The White Castle*

Este juego combina mecánicas de colocación de trabajadores y manejo de recursos, ambientado en la construcción y administración de un castillo japonés. Los jugadores compiten por optimizar su estrategia y obtener la mayor cantidad de puntos de victoria.

Recursos sobre el juego:

- **BoardGameGeek**: Plataforma principal con información detallada sobre el juego, estadísticas, reglas, reseñas y foros de discusión.
  Enlace: https://boardgamegeek.com/boardgame/
  Secciones clave:

    Files: Archivos con manuales, hojas de ayuda y resúmenes de reglas.

    Forum: Discusión de reglas, estrategias y aclaraciones de dudas.

    Stats: Estadísticas que pueden ser usadas para análisis tabular.

- **Análisis en blogs especializados**: Se utilizaron entradas de blogs con reseñas y análisis sobre *The White Castle*.
- **Videos de reseñas**: Plataforma como YouTube ofrece revisiones del juego, disponibles en varios idiomas. Se consiguió subtítulos de varios videos para enriquecer las fuentes textuales.
