



Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Tecnicatura en Inteligencia Artificial

Procesamiento del Lenguaje Natural

Trabajo Práctico Final.

Alumnos: Menescaldi, Brisa M-7184/6

Docentes: Ferrucci, Constantino
Geary, Alan
Manson, Juan Pablo
Sollberger, Dolores

ÍNDICE

Introducción ----- pág. 3

RAG

Resumen ----- pág. 4

Desarrollo ----- pág. 5 - 14

Conclusiones ----- pág. 15

Modelos y Librerías utilizadas ----- pág. 16 - 17

Agente

Resumen ----- pág. 18

Desarrollo ----- pág. 19 - 21

Conclusiones ----- pág. 22

Modelos y herramientas utilizadas ----- pág. 23

Introducción

El trabajo realizado consistió en la realización de un chatbot experto en el juego de mesa 'The White Castle' primero usando la técnica de RAG (Retrieval Augmented Generation) y luego la incorporación de un Agente, basado en el concepto ReAct. Nos vamos a enfocar en que mediante estas herramientas, el usuario pueda adquirir conocimientos sobre reglas del juego pero también información estadística de este e información sobre otros juegos publicados por los autores del mismo.

Como objetivo principal se buscó que se pueda interpretar la consulta del usuario y poder ir a buscar en las bases de datos brindadas logrando una respuesta que tenga sentido y sin la necesidad de buscar información fuera de estas.

Durante este informe vamos a abordar las metodologías usadas durante el desarrollo de los mismos.

RAG

Resumen

Este desarrollo del chatbot enfocado en el juego de mesa 'The White Castle' haciendo implementación de un RAG, haciendo sus etapas desde el prompt del usuario, clasificación, retriever para obtener información de nuestras distintas base de datos como lo son la base de datos vectorial, la base de datos de grafos y los datos tabulares y por último hacer uso de un LLM para obtener una respuesta considerando a la pregunta realizada.

Desarrollo

Como entorno de trabajo se debió hacer uso de un entorno de Google Colab, en el cuál se debió hacer uso de las siguiente librerías:

para la realización de **web-scrapping**; `selenium` para traer información de páginas web dinámicas generadas por JavaScript, `webdriver-manager` ayuda a la librería anterior a controlar los navegadores, `requests` para realizar consultas HTTP y `beautifulsoup4` para trabajar con HTML. Y de esta manera, poder hacer extracción de información que se encuentra disponible en distintas páginas web. A su vez, como extracción de datos, se utilizó `youtube-transcript-api` para obtener transcripciones (subtítulos) de los videos disponibles en Youtube.

para la realización de la **base de datos vectorial**; `sentence_transformers` para generar los embeddings de los textos, `chromadb` para poder allí almacenar y consultar nuestros datos basados en representaciones vectoriales (embeddings) y así poder tener recuperación de información. Y haciendo relación a la recuperación de información, `langchain` para particionar (split) cadenas de texto, `rank_bm25` haciendo un ranking para recuperar la información más relevante en una consulta como lo son las palabras claves, y `nltk` ayuda a preprocesar el texto antes de calcular su relevancia, tokenizando y normalizando la consulta.

para la realización de la **base de datos de grafos**; `redis` nos crea un servidor a donde consultar la base de datos de grafos usando Cypher, `redisgraph` para crear la base de datos de grafos, `networkx` para crear redes de grafos y `matplotlib` para visualizar el grafo.

para **clasificador**; también se hizo uso de `sentence_transformers`, `transformers` se vincula con la librería anterior para utilizar modelos de aprendizaje profundo pre entrenados, `scikit-learn` para poder implementar el modelo de clasificación.

y como librerías que no requieren instalación; `numpy` para poder trabajar con arreglos de números, `huggingface_hub` para poder utilizar modelos de Hugging Face mediante tokens, `re` para poder utilizar expresiones regulares, `pandas` para generar DataFrames.

Para las fuentes de datos, se generaron tres fuente de datos para poder darle contexto a las preguntas del usuario:

- **Base de datos tabulares:** vamos a contar con un DataFrame el cuál tiene información estadística del juego sacada mediante web scrapping de la página web BGG. El cuál contiene las métricas: Promedio de calificación, Número de Calificaciones, Desviación estándar, Complejidad, Comentarios, Aficionados, Vistas de la página; y sus respectivos valores.

Dónde cada métrica indica lo siguiente:

Promedio de calificación, puntaje promedio dado por los usuarios del sitio web BGG.

Número de Calificaciones, votantes para la puntuación del juego.

Desviación estándar, desviación estándar del puntaje del juego.

Complejidad, nivel de dificultad en una escala del 1 al 5.

Comentarios, cantidad de comentarios respecto al juego en dicho sitio web.

Aficionados, fans del juego.

Vistas de la página, cantidad de visitas en la página BGG sobre este juego.

Para poder hacer la extracción de estos datos, se prepara el sistema para la utilización de 'selenium' haciendo uso de esta librería debido a que contamos con una página dinámica. Aparte de importar lo necesario de dicha librería, se configura un entorno automatizado para interactuar con un navegador Chrome como la eliminación de la interfaz gráfica entre otras para asegurar que 'selenium' funcione correctamente en entornos con recursos limitados.

Una vez que se tiene esto configurado, procedemos a extraer información sobre el juego de mesa 'The White Castle', desde la página de BoardGameGeek (BGG). De esta se extraen métricas, considerando un lapso de 20 segundos para poder cargar el contenido de la página, allí se localiza el primer que contiene las métricas del juego utilizando un selector CSS específico y dentro de esta se buscan cada ya que dentro se encuentran los títulos (Métrica) y su respectiva descripción (Valor) a extraer, los cuales almacenamos en una lista. Como punto a mejorar, lo que considero como métrica, lo cuál es el título, dentro de la página se encuentra en inglés, la traducción en mi caso la hago de forma manual pero lo ideal sería hacer uso de la traducción de texto dada en teoría como GoogleTranslator.

Con esto, luego, se crea un DataFrame de Pandas y se guarda como un archivo CSV llamado "estadisticas_juego.csv".

	Métrica	Valor
0	Promedio de calificación	7.980
1	Número de Calificaciones	11,599
2	Desviación estándar	1.19
3	Complejidad	3.02 / 5
4	Comentarios	1,597
5	Aficionados	1,829
6	Vistas de la página	1,360,234

Como ya dijimos, una etapa de la implementación de un RAG es el uso de retrieve el cuál busca información de la base de datos, se define una consulta dinámica para que se vaya a buscar la información que se necesite dentro de nuestros datos tabulares, para eso mediante la función definida `tabular_dinamica(imput)` la cuál usa una API de Hugging Face para interactuar con un modelo de lenguaje (QWen 2.5) el cuál interpreta una consulta del usuario sobre estadísticas del juego y generar una consulta de base de datos que se puede usar para recuperar esa información, del cuál luego ejecutamos para obtener la respuesta como texto. Podemos ver el siguiente ejemplo de lo que nos devuelve nuestra consulta dinámica para una consulta en específica.

```
eval(tabular_dinamica('¿Qué promedio de calificaciones tiene?'))  
'7.980'
```

Como observación IMPORTANTE a considerar, precisamente en lo que es nuestro web scraping, para la obtención de datos que utilizaremos para crear nuestro DataFrame, funciona al correrlo en un entorno como Colab con CPU. Ya que al intentar realizarlo con la GPU que nos ofrece dicho servicio puede acceder y localizar los elementos pero no extrae la información que estos tienen.

- **Base de datos vectoriales:** para obtener la documentación, vamos a extraer contenido desde una página web no teniendo en cuenta las imágenes y una transcripciones de dos videos de YouTube, siendo que del primer video se filtra el texto hasta el minuto 35:25 y el segundo video se filtra el texto entre los segundos 48 y 11:46. Una vez obtenido, se pasa todo a minúscula.

```
---- Artículo ----  
Japón, 1761. Provincia de Harima. El Daimio Sakai Tadazumi es uno  
=====  
---- Video 1 ----  
existen diversos juegos de mesa que pueden ser definidos como comp  
=====  
---- Video 2 ----  
preparación preparación del Castillo colocamos el tablero y cogemo
```

Procedemos a la segmentación del texto, donde se usa `RecursiveCharacterTextSplitter` para dividir el texto completo en "trozos" de tamaño manejable, con un tamaño de bloque de 500 caracteres y una superposición de 50 caracteres entre bloques consecutivos y a estos fragmentos de textos se le indica una metadata el cuál contiene la fuente de dónde se sacó dicho fragmento.

```

Metadata: reseña
teniendo un espacio libre bastante amplio en la zona izquierda de

=====

Chunk 66: 441 caracteres
Metadata: reseña
Pero con todo, es un juego muy satisfactorio cuando se consigue h

=====

Chunk 67: 494 caracteres
Metadata: tutorial-aprender a jugar
existen diversos juegos de mesa que pueden ser definidos como comp

=====

Chunk 68: 495 caracteres
Metadata: tutorial-aprender a jugar

```

Nuestro objetivo va a ser vectorizar estos chunks (fragmentos de texto) convirtiéndolo en embeddings, los cuáles realizamos con un modelo pre entrenado de Sentence-BERT el cual genera las incrustaciones necesarias para que luego podamos realizar búsquedas o comparaciones semánticas. Posteriormente, guardamos nuestros chunks, con su respectiva metadata, su respectivo ID de identificación y su incrustación (embeddings) en una base de datos vectorial ChromaDB.

```

Documentos almacenados en ChromaDB:
ID: chunk_1, Texto: japon, 1761. provincia de harima. el daimio sakai ..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 1}
ID: chunk_2, Texto: cuidan hasta el último detalle de los jardines de ..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 2}
ID: chunk_3, Texto: de 12 añosy unaduración aproximada de 80 minutos. ..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 3}
ID: chunk_4, Texto: y prefieren hacerse una idea general de cómo funci..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 4}
ID: chunk_5, Texto: a lo largo de tres eras con tres rondas por era pa..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 5}
ID: chunk_6, Texto: de acciones consiste en unos dados en tres colores..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 6}
ID: chunk_7, Texto: puntos por la posición de cada uno de sus peones. ..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 7}
ID: chunk_8, Texto: daimio, un espacio de colocación de dado con un va..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 8}
ID: chunk_9, Texto: orden de turno y el track de influencia (dividido ..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 9}
ID: chunk_10, Texto: pasarela. esto es, el de mayor o menor valor de un..., Metadata: {'categoria': 'reglas del juego', 'chunk_index': 10}

```

Considerando que una etapa de la implementación de un RAG es el uso de retrieve el cuál busca información de la base de datos, se define una búsqueda híbrida de la cuál se buscan las palabras claves, se hace una búsqueda semántica y con la unión de estas dos se realiza un re ranking de lo obtenido.

A través de los chunks que tenemos en nuestra base de datos vectoriales, vamos a hacer uso de estos para que mediante la implementación de técnicas podamos recuperar la información que necesitamos. Principalmente vamos a tokenizar los textos y la consulta a español para poder realizar la comparación correspondiente mediante `word_tokenize` de NLTK , luego con el objetivo de obtener las palabras claves se utiliza el algoritmo BM25 para encontrar lo más relevantes a partir de una consulta y posteriormente para realizar una búsqueda semántica se convierte la consulta a embeddings, los cuáles busco en mi base vectorial para de allí obtener lo que se encuentre con mayor similitud. Finalmente, se realiza el re-ranking combinando los resultados obtenidos en BM25 de palabras claves y en la búsqueda semántica, y obteniendo como respuesta los documentos ordenados según la relevancia combinada y así podemos obtener mejores resultados en la tarea de búsqueda.

Podemos obtener a través de una consulta, los resultados que obtenemos al buscar palabras claves y una búsqueda semántica y rankear dichos resultados.


```

query = "Qué contiene el tablero principal?"

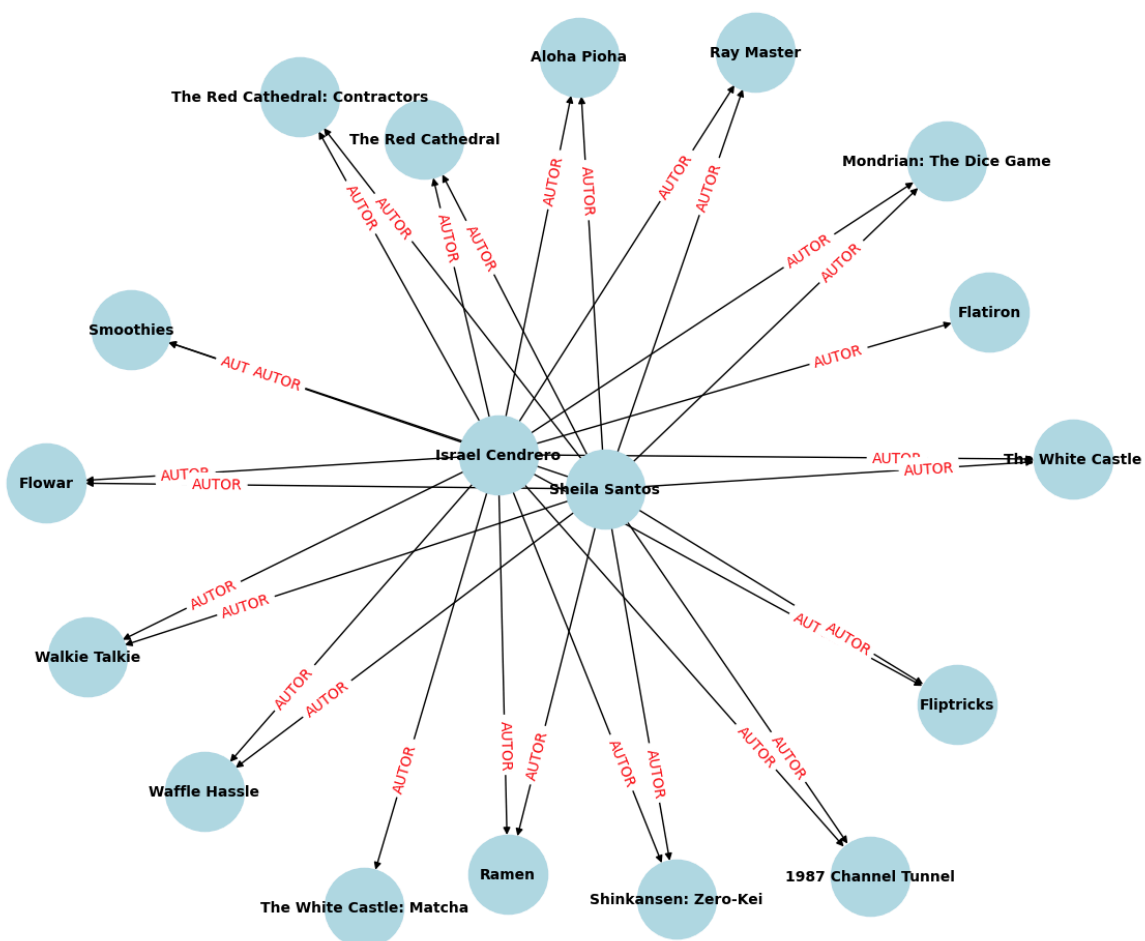
resultados = re_rank(query, lista_chunks, collection, top_n=5)
print("Resultados rerankeados:", resultados)

Resultados rerankeados: ['en un espacio de acción disponible (en el tablero principal o en el personal). a la hora de ocupar el espacio de a

```

- **Base de datos de grafos:** como primer paso, procedemos a lo que es la creación de nuestro grafo para eso se establece conexión con el servidor Redis y se crea un grafo usando RedisGraph para almacenar información sobre autores y sus juegos y que para cada uno de ellos se crean nodos y se establecen relaciones entre los nodos de tipo "AUTOR" para indicar qué autor ha creado qué juego. Posteriormente para poder realizar consulta en RedisGraph se utiliza Cypher siendo un lenguaje de consulta de grafos donde estas consultas buscan todas las relaciones existentes entre autores y juegos y la cuál devuelve tanto los nombres de los autores como los juegos asociados a cada uno. Y mediante networkx, podemos representar las relaciones, obteniendo este de visualmente.

Visualización del grafo de autores y juegos



Así como realizamos una consulta dinámica para nuestra base de datos tabulares a través de la consideración que una etapa de la implementación de un RAG es el uso de retrieveer el cuál busca información de la base de datos, en este caso también se define una búsqueda híbrida que mediante la definición de la función

`grafos_dinamica(query, graph)` que a través de la conexión a la API del modelo de lenguaje QWen 2.5, donde se le pide al modelo que reciba una consulta del usuario y la convierta en una consulta Cypher válida indicando las entidades y relaciones entre ella que contiene nuestro grafo. La consulta generada, se ejecuta en el grafo utilizando el `graph.query()` y finalmente devolviendo el resultado de una forma legible.

Podemos ver la implementación de dicha consulta dinámica, obteniendo ya nuestro resultado legible y no es lenguaje de consulta Cypher, tanto para una consulta sobre autores de un juego o juegos de un autor.

```
print(grafos_dinamica('Quién es el autor de "Flatiron"?', graph))
```

```
Consulta Cypher generada: MATCH (a:Autor)-[:AUTOR]->(g:Juego {nombre: 'Flatiron'}) RETURN a.nombre
Resultados:
Israel Cendrero
```

```
print(grafos_dinamica('Que juegos tiene el autor "Israel Cendrero" ?', graph))
```

```
Consulta Cypher generada: MATCH (a:Autor {nombre: 'Israel Cendrero'})-[:AUTOR]->(g:Juego) RETURN g.nombre
Resultados:
Aloha Pioha
Mondrian: The Dice Game
Ray Master
Ramen
1987 Channel Tunnel
Smoothies
Waffle Hassle
Fliptricks
The Red Cathedral
Shinkansen: Zero-Kei
Walkie Talkie
The Red Cathedral: Contractors
Flowar
The White Castle
Flatiron
The White Castle: Matcha
```

Clasificadores

Como punto principal, nuestro objetivo es poder clasificar de forma correcta las consultas que el usuario ingrese para que una vez identificado se pueda acceder a la fuente de datos correspondiente a hacer búsqueda de información y así poder brindar una respuesta.

Siendo,

“Autores” → Base de datos de grafos

“Estadísticas del Juego” → Base de datos tabulares

“Reglas del Juego” → Base de datos vectorial

Se desarrollaron dos versiones de clasificadores;

- modelo entrenado con ejemplos y embeddings:

Vamos a entrenar un modelo de clasificación de texto que asigna una de las tres categorías "Autores", "Estadísticas del Juego" y "Reglas del Juego" donde se usa modelo pre entrenado de Sentencia Transformer para transformar las frases de entrada en vectores, primero se le proporciona un conjunto de datos con ejemplos de frases asociadas a las categorías mencionadas, dividiendo dicho conjunto en entrenamiento (80%) y en prueba (20%), posteriormente se crea un clasificador de Regresión Logística Multinomial para realizar las predicciones y se entrena el modelo con los datos vectorizados. Para evaluar dicho modelo, se realizan las predicciones sobre el conjunto de prueba, se calcula la precisión del modelo siendo el porcentaje de predicciones correctas, y una evaluación de distintas métricas como recall, precisión y F1 para cada categoría.

Obteniendo lo siguiente,

```
Precisión Regresión Logística: 1.0
Reporte de clasificación:
              precision    recall  f1-score   support

     0           1.00        1.00        1.00         2
     1           1.00        1.00        1.00         2
     2           1.00        1.00        1.00         2

 accuracy                   1.00         6
 macro avg           1.00        1.00        1.00         6
 weighted avg        1.00        1.00        1.00         6
```

Pudiendo observar que el 100% de las predicciones fueron correctas y que el modelo clasificó de forma correcta todas las clases. Pero como punto a considerar, cada clase tiene solamente 2 ejemplos en el conjunto de prueba lo cuál puede ser muy pobre. Nuestro conjunto de datos en el cual se entrena es muy pequeño y entonces el modelo no ha sido probado en una variedad de datos, lo cuál puede realizar un overfitting en donde el modelo podría no generalizar tan bien con datos nuevos si estos son muy diferentes de los ejemplos entrenados. Entonces, como punto a mejorar sería la aumentación del conjunto de datos.

También podemos ver que tiene un buen comportamiento para consultas nuevas que le paso, clasificando de forma correcta.

```
Texto: 'Israel Cendrero diseñó el juego 'Smoothies'.'  
Clasificación predicha: Autores  
  
Texto: 'Este juego tiene una puntuación promedio de 8.2.'  
Clasificación predicha: Estadísticas del Juego  
  
Texto: 'Los jugadores deben maximizar sus recursos para ganar.'  
Clasificación predicha: Reglas del Juego
```

- LLM

También buscamos clasificar en una de las siguientes categorías "Autores", "Estadísticas del Juego" y "Reglas del Juego" pero en este caso, usando un modelo de lenguaje de Hugging Face (QWen 2.5). Para este, tenemos la posibilidad de la construcción de un prompt para allí indicarle cómo debe comportarse nuestro modelo, como clasificar la consulta ingresada en alguna de las 3 categorías con las que contamos, siendo un detalle importante que el prompt que le ingresamos al modelo está diseñado para que el modelo solo devuelva la categoría correspondiente, sin ningún contexto adicional ni escribir la respuesta. Y también se le especifican ejemplos para darle una noción de cómo este debe actuar. Justamente por estas razones, particularmente la información y aclaraciones que puedo sumarle al prompt, es que elijo utilizar como modelo de clasificación LLM; a pesar de haber obtenido buenas métricas en nuestro modelo de regresión lineal pero del cual tenemos una pobreza en nuestro conjunto de datos.

Al probarlo con distintas consultas, observamos una perfecta clasificación:

```
clasificador_llm("Quién es el autor de el juego 'Smoothies'?")  
  
'Autores'  
  
clasificador_llm("Cuál es la puntuación promedio del juego?")  
  
'Estadísticas del Juego'  
  
clasificador_llm("Qué deben hacer los jugadores para ganar?")  
  
'Reglas del Juego'
```

Retrieval Augmented Generation(RAG)

Para lo que es la implementación del sistema RAG, se le solicita al usuario que ingrese una consulta y en caso de querer salir, escribir 'salir'. Una vez obtenida la consulta, se utiliza la función `clasificador_llm(query)` ya definida, la cuál clasifica la consulta ingresada y nos imprime en pantalla, luego basándonos en la categoría clasificada vamos a ir en busca del contexto (retriever) ya que el sistema elige un método de recuperación de información apropiado para extraer el contexto relevante siendo la función `re_rank` para buscar información en un conjunto de fragmentos de texto (`lista_chunks`) almacenados en la base de datos vectoriales o la función `grafos_dinamica` para recuperar información sobre los autores utilizando un gráfico dinámico o la función `tabular_dinamica` que recupera estadísticas basadas en datos tabulares.

Se hace uso del modelo pre entrenado QWen 2.5, donde se le prepara un prompt donde se define el comportamiento del bot donde principalmente se le pide que debe responder sólo en función del contexto recuperado. Por último, utiliza el contexto y la consulta para generar una respuesta que se encuentra basada exclusivamente en las fuentes de datos brindadas.

```
RAG()
Ingrese una consulta (o 'salir' para terminar): como se inicia el juego?
Categoría Predicha: Reglas del Juego
Para iniciar el juego de The White Castle, sigue estos pasos:
1. Coloca el tablero principal en el centro de la mesa.
2. Asegúrate de que las zonas del tablero estén bien formadas, incluyendo el puente, los espacios de acción, y las áreas de recursos.
3. Toma la ficha de marcador de ronda y colócala en el número 1.
4. Cada jugador debe ubicar su ficha de influencia al principio de la garza. Esta zona determina el orden de juego, es decir, quién inicia y quién sigue.
5. Si estás jugando con tres jugadores, utiliza las fichas de colores amarillo, rojo y azul. Colócalas en la zona de garzas para indicar el orden de juego.
Una vez que todos los jugadores estén listos y las fichas estén colocadas correctamente, el juego puede comenzar. El jugador que esté más arriba en el marcador de
Ingrese una consulta (o 'salir' para terminar): que nivel de complejidad tiene el juego?
Categoría Predicha: Estadísticas del Juego
Según la información proporcionada en la ficha técnica del juego (3.02 / 5), el nivel de complejidad de The White Castle es moderado. Sin embargo, esta calificación
Ingrese una consulta (o 'salir' para terminar): quienes son los autores de The White Castle?
Categoría Predicha: Autores
Consulta Cypher generada: MATCH (a:Autor)-[:AUTOR]->(g:Juego {nombre: 'The White Castle'}) RETURN a.nombre
No tengo información para responder tu pregunta.
Ingrese una consulta (o 'salir' para terminar): salir
Saliendo del sistema...
```

Nos encontramos que al momento de hacerle la consulta al RAG basada en autores/juegos, nos clasifica de forma correcta la categoría “Autores” y hasta nos da una correcta consulta Cypher pero por alguna razón que no logro identificar nos dice que no tiene información para responder nuestra consulta sobre dicho tema, dando a entender que no puede procesar en Redis la consulta.

```
[ ] consulta= graph.query("MATCH (a:Autor)-[:AUTOR]->(g:Juego {nombre: 'The White Castle'}) RETURN a.nombre")
print(consulta.result_set)

[ ] print(grafos_dinamica('Quienes son los autores de The White Castle?', graph))

Consulta Cypher generada: MATCH (a:Autor)-[:AUTOR]->(g:Juego {nombre: 'The White Castle'}) RETURN a.nombre
Resultados:
Israel Cendrero
Sheila Santos
```

A su vez, podemos ver como podemos procesar la consulta con Redis en base a la consulta Cypher que nos da el RAG y nos trae de forma perfecta la respuesta. Como también si ejecutamos la consulta que le pasamos a la función nos imprime exactamente la misma consulta y trayendo la respuesta que le corresponde.

A pesar de realizar nuestras pruebas, gracias a la ayuda de un docente, pudimos identificar que el error venía de querer pasarle a nuestra función de RAG en el caso de nuestra base de grafos dinámica como contexto directamente la respuesta. Realizando así una modificación a nuestra consulta dinámica de la base de grafos al momento de implementarla en nuestro RAG, considerando en ese caso como contexto solamente la parte en la que obtengo la consulta que se le debe hacer a mi base de grafos.

Obteniendo así un excelente desempeño y buenas respuestas de parte de mi RAG para todas las categorías y las fuentes de datos.

```
RAG()
```

```
Ingrese una consulta (o 'salir' para terminar): como se inicia el juego?
```

```
Consulta ingresada: como se inicia el juego?
```

```
Categoría Predicha: Reglas del Juego
```

```
Para iniciar el juego de The White Castle, se deben seguir estos pasos:
```

1. Coloca el tablero principal en el centro de la mesa.
2. Asegúrate de que el marcador de ronda esté en la posición número 1.
3. Cada jugador debe ubicar su ficha de influencia al principio de la garza. Esta zona determina el orden de juego,
4. Si estás jugando con tres jugadores, utiliza las fichas de colores amarillo, rojo y azul, y colócalas en la zona

```
Estos pasos aseguran que el juego comience de manera organizada y que todos los jugadores estén listos para su turno
```

```
Ingrese una consulta (o 'salir' para terminar): que nivel de complejidad tiene?
```

```
Consulta ingresada: que nivel de complejidad tiene?
```

```
Categoría Predicha: Estadísticas del Juego
```

```
El nivel de complejidad de The White Castle es moderado, con una calificación de 3.02 sobre 5. Esto indica que el j
```

```
Ingrese una consulta (o 'salir' para terminar): quienes son los autores de 'Ramen'?
```

```
Consulta ingresada: quienes son los autores de 'Ramen'?
```

```
Categoría Predicha: Autores
```

```
Los autores de 'Ramen' son Israel Cendrero y Sheila Santos.
```

```
Ingrese una consulta (o 'salir' para terminar): salir
```

```
Consulta ingresada: salir
```

```
Saliendo del sistema...
```

Conclusiones

Podemos concluir que nos encontramos frente a la utilización de la técnica RAG para la realización de nuestro chatbot sobre el juego de mesa asignado se obtuvo un gran y excelente desempeño ya que contamos con un buen clasificador donde nos clasifica de la forma correcta para de allí saber a dónde ir a buscar nuestro contexto para sacar la información, también habiendo realizado que nuestras fuentes se encuentren mayormente automatizadas tanto para la extracción de fuentes como su búsqueda y consultas de forma dinámica.

Como puntos a mejorar para seguir con esta automatización, es la obtención de los datos de la base de grafos mediante el uso de una técnica de extracción de texto como web scraping o mediante algún archivo .pdf y otra cosa a mejorar es en los datos tabulares, considerando que contamos con un sitio web en inglés, que luego de obtener la extracción de los valores hacer uso de alguna técnica que nos tradujera. Ya que con estas mejoras, tendríamos un excelente bot totalmente automatizado.

Modelos y librerías utilizadas

Qwen/Qwen2.5-Coder-32B-Instruct → HuggingFace

<https://huggingface.co/Qwen/Qwen2.5-Coder-32B-Instruct>

huggingface_hub

https://huggingface.co/docs/huggingface_hub/index

Selenium

<https://www.selenium.dev/>

webdriver-manager

<https://pypi.org/project/webdriver-manager/>

requests

<https://pypi.org/project/requests/>

beautifulsoup4

<https://pypi.org/project/beautifulsoup4/>

chromadb

<https://www.trychroma.com/>

youtube-transcript-api

<https://pypi.org/project/youtube-transcript-api/>

re

<https://docs.python.org/3/library/re.html>

pandas

<https://pypi.org/project/pandas/>

langchain.text_splitter

<https://pypi.org/project/langchain-text-splitters/>

sentence_transformers

<https://pypi.org/project/sentence-transformers/>

rank_bm25

<https://pypi.org/project/rank-bm25/>

nltk

<https://pypi.org/project/nltk/>

numpy

<https://pypi.org/project/numpy/>

redis y redisgraph

<https://pypi.org/project/redis/>

networkx

<https://pypi.org/project/networkx/>

matplotlib

<https://pypi.org/project/matplotlib/>

transformers

<https://huggingface.co/docs/transformers/index>

scikit-learn

<https://scikit-learn.org/stable/>

Agente

Resumen

En este caso, en el desarrollo del chatbot enfocado en el juego de mesa 'The White Castle' haciendo implementación de un Agente basándonos especialmente en el concepto ReAct. Pudiendo hacer una reutilización en casi su mayoría de las funciones definidas para nuestro RAG como las fuentes de datos y nuestros retrievers.

Como beneficio, un agente puede identificar cuál función (en este caso, denominadas herramientas) debe hacer uso en base a la consulta que el usuario le ingrese, interpretar los resultados de esta y devolver una respuesta acorde. Y también tiene la posibilidad de iterar por distintas herramientas por una cantidad de interacciones hasta conseguir la respuesta adecuada o en su defecto devuelve un error.

Desarrollo

Dentro de nuestro entorno de Google Colab, primero se probó el uso de modelos como el presentado a modo práctico en clases `phi3` pero nos topamos con errores de conexión a este para poder realizar la implementación de nuestro agente, por lo tanto también se probó `lama3.2:latest` siendo este el modelo con el que mejores resultados se obtuvieron, y luego de reiteradas pruebas nos quedamos con la última solución obtenida.

Por esa razón se procede a la instalación del último modelo de Ollama.

Se realiza la definición de las herramientas/tools del agente, en dónde se hace uso de los retrievers utilizados anteriormente para la implementación del sistema RAG, los cuales tienen una conexión directa con nuestra base de datos.

- `doc_search` obtiene una consulta y da los mejores documentos rankeados de la base vectorial.

- `graph_search` vamos a obtener como respuesta la consulta a la base de grafos, luego de obtener la query correspondiente a esta.

- `table_search` realiza lo mismo en donde se obtiene la respuesta de la consulta a la base tabular, luego de obtener la query correspondiente a esta.

Al realizar nuestro agente, se le establecieron diversos parámetros, entre ellos; Darle una temperatura muy baja para que al momento de realizar la búsqueda se centre en la información de contexto que tiene y solamente responda con las herramientas brindadas. Se le especifica que debe devolver respuestas pocas extensas y para poder responder estas que la información que reciba sea corta. Realización de un prompt el cuál se le informa a nuestro modelo que se encuentra diseñado para responder preguntas solamente sobre el juego 'The White Castle' usando solo tres herramientas: `graph_search` (para autores), `table_search` (para estadísticas del juego) y `doc_search` (para reglas del juego). Cada consulta debe analizar para determinar qué herramienta utilizar, y debe usar cada una solo una vez, siguiendo las instrucciones temáticas. Debe responder de manera concisa, combinando resultados de las herramientas según sea necesario, sin modificar la consulta recibida. A su vez, también dando ejemplos de cómo debe actuar e interactuar hasta obtener la respuesta acertada.

Una vez ya definido lo anterior, se procede a interactuar el agente con el usuario, obteniendo los siguientes resultados:

- Consulta 1: ¿Cómo se prepara cada partida?

Respuesta 1: Para prepararse cada partida, los jugadores deben seguir estos pasos: 1) Se recogen todos los dados de la primera y segunda ronda, se tiran y se colocan de nuevo sobre los puentes siguiendo la regla indicada en la preparación de la partida. 2) Se avanza la ficha de ronda una posición y comienzan los turnos de jugador final de partida. 3) Después de reordenar el orden de turno, se cuentan los puntos finales y los puntos obtenidos durante la partida. Los jugadores también

obtienen recursos sobrantes por cada carta que colocan en su hilera de bonificaciones de lámpara.

Obtuvimos dicha respuesta siendo que primero consultó en la base tabular y posteriormente si fue a nuestros datos vectoriales que es allí en donde se encontraba nuestra información.

- Consulta 2: ¿Con qué cantidad de aficionados cuenta el juego?

Respuesta 2: El juego de ajedrez cuenta con entre 2 y 4 aficionados.

En este caso, la respuesta no es la esperada, a pesar que en un principio intenta buscar información de nuestra base tabular siendo que allí es donde se podía encontrar la respuesta correcta, luego indaga en la base de grafos y posteriormente en la base vectorial que es de donde trae la respuesta final considerando una coincidencia de palabra clave 'aficionados'.

- Consulta 3: ¿Qué comentarios existen sobre el juego y cuánta cantidad de comentarios hay ?

Respuesta 3: El juego "The White Castle" es considerado gratificante y enriquecedor, ya que permite al jugador ver cómo su estrategia se desarrolla. También es un juego desafiante, ya que cada turno es importante y resolverlo de forma "simple" puede no ser lo más efectivo.

Para obtener esta respuesta nuestro agente, primero indaga sobre la base tabular, siguiendo por la base de vectorial, volviendo a la base tabular, siguiendo nuevamente por la base vectorial, luego indagando en la base de grafos, base tabular y finalmente en la base vectorial. Podemos ver un recorrido y en más de una ocasión por todas nuestras herramientas. Nuestro objetivo con esta consulta era que pueda traer información tanto desde la base vectorial como de la base tabular pero a pesar de eso, nuestra respuesta final solamente contiene información de la base vectorial.

A pesar de esto, desde la primera consulta a la herramienta ya nos realiza un cambio en la query vinculando con el juego de mesa 'Ajedrez'.

- Consulta 4: Los autores del juego 'The White Castle', ¿también tienen otros juegos?

Respuesta 4: Error al procesar la consulta: Reached max iterations.

Nos encontramos con un error en donde se supera el máximo de interacciones sobre nuestras herramientas al momento de obtener nuestra respuesta, la idea era obtener información de la base de grafos pero el agente hizo una reiteradas búsqueda a las base tabulares y finalmente antes de quedarse sin iteraciones busca en la base vectorial, nunca llegando a la fuente de datos en donde sí se podría haber sacado la respuesta que buscábamos.

A pesar de esto, como punto a favor es que nuestra query nunca fue modificada en las iteraciones por las diferentes herramientas.

- Consulta 5: ¿Qué hace el jugador en cada turno del juego?

Respuesta 5: Error al procesar la consulta: Reached max iterations.

Nos topamos con el mismo problema que en la consulta anterior pero en este caso idea era obtener data de la base de vectorial y el agente recorre base tabular, base tabular y particularmente en este caso nos hace un cambio de query **'¿Cómo se**

prepara cada partida en el juego de ajedrez?' donde claramente como última búsqueda va a la base de grafos y a pesar de obtener la consulta en

lenguaje Cypher **Consulta Cypher generada: MATCH**

(a:Autor)-[:AUTOR]->(g:Juego {nombre: 'Ajedrez'}) RETURN a.nombre

pero le ingresa el juego de 'Ajedrez' que no se encuentra definido.

Conclusiones

A pesar que el uso de agentes tiene la particularidad de ser más general y tener un enfoque en la autonomía siendo un sistema autónomo principalmente en la toma de decisiones en base al contexto, pero nos topamos con dos grandes inconvenientes ya que tiene dificultades para mantener la consulta del usuario intacta y pasarla correctamente como parámetro a las herramientas, lo que hace que se generen errores o haya una falta de obtención de información. Además, no siempre identifica correctamente la base de datos a utilizar, a pesar de las indicaciones de seguir un orden de preferencia en función de la relevancia indicada en el prompt definido. Por esa razón las respuestas obtenidas no siempre suelen ser las adecuadas.

Como puntos a mejorar sería hacer una verificación más manual de las query que va interactuando en las herramientas para una misma consulta para evitar que esta sea modificada. Y luego buscar la manera de primero poder seleccionar la herramienta más adecuada y de mayor relevancia para la consulta realizada por el usuario.

Modelo y herramientas utilizadas

Ollama

<https://ollama.com/library/llama3.2>

LLama-Index

<https://docs.llamaindex.ai/en/stable/>