Trabajo Práctico N.º 2

Materia: **IA4.2 Procesamiento Del Lenguaje Natural**
Carrera: **Tecnicatura Universitaria en Inteligencia Artificial**

Alumno: **Ignacio Eloy González: G-5933/1**

Profesores: **Alan Geary, Juan Pablo Manson.**

Fecha: 18/12/2024

Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Índice

Introducción

El primer ejercicio de este trabajo consta de crear un chatbot experto en un juego de mesa designado por la cátedra, donde para que nuestra LLM que utilizamos como chatbot sepa del juego de mesa en cuestión, implementaremos un RAG el cual le proporcionará el contexto necesario para que la misma pueda responder. Para eso, lo primero fue crear una base de datos vectorial, la cual tendrá la información textual del juego como reglas, reseñas, guías estratégicas y más. Para esta base fue necesario extraer información de varias páginas de reseñas utilizando web scrapping, también en la página boardgamegeek dentro del apartado files del juego de mesa, se encontraban la mayoría de los pdfs utilizados, que por cuestión de practicidad se descargaron manualmente y se guardaron en una carpeta de drive, junto con el texto extraído de las reseñas. Una de las reseñas estaba en español, por lo que fue necesario traducir la misma, este proceso consto de separar los párrafos por sus “\n” o newlines para traducir con GoogleTranslate, en específico la librería deep-translator. Una vez obtenidos todos los archivos de texto en la carpeta de Google Drive, se procedió a descargarlos dentro del contorno de Colab para así ya poder empezar a limpiar el texto e introducirlo en la base de datos vectorial, la cual se creó en chromaDB. La limpieza del texto constó de reemplazar ciertos caracteres encontrados por la librería de obtención de texto de los pdfs, quitar espacios blancos, detrás y adelante de los párrafos, documentos y oraciones. Para ingresar el texto en la base es necesario dividir el texto en diferentes fragmentos para así poder recuperar información más exacta. Para generar estos fragmentos se utilizó un splitter de Langchain el cual utiliza un modelo de embeddings para separar el texto basándose en una similitud semántica, donde cada recorte del texto contiene un sentido semántico. Los resultados fueron bastante concisos, ya que ningún recorte es un párrafo cortado en la mitad o cualquier lugar donde no sea un punto. Luego de crear todos los “chunks” de los documentos, fue necesario encontrar los embeddings de estos chunks con un modelo, el cual utilice en un trabajo práctico anterior que tuvo un desempeño muy bueno. Para que a la hora de hacer una búsqueda semántica en la base de datos, con un embedding de la query con este mismo modelo, la búsqueda funcione de manera correcta. Se ingresa toda la información en la base de datos vectorial y esta ya está lista para ser utilizada.

La siguiente base de información fue la base de datos de grafos, en donde se utilizó RDF, ya que me pareció que SPARQL tenía un formato de querys bastante bueno y no tan complicado de usar. Para esta base de grafos, fue necesario extraer diferente información de diseñadores, artistas, publicadores, premios, relación entre los diseñadores, premios de los diseñadores, otros juegos de los diseñadores y artistas, lugar donde se encuentra el juego, religión del juego, entre otros. Toda esta información fue extraída de la página boardgamegeek proporcionada por la cátedra, con web scrapping, en específico Selenium. Luego se ingresó la información en la base de grafos, utilizando como nodo central o principal el juego en sí y en todas las aristas la relación entre el juego y las entidades antes nombradas.

Por último, la base de datos tabular que se creó en pandas, para ser exportada como un archivo CSV. Para esta base se buscaron ingresar datos mayormente estadísticos o numéricos, así como año de lanzamiento del juego, edad recomendada mínima para jugar al juego, tiempo de juego, número promedio de rating, jugadores totales, ranking, etc. Toda esta información se obtuvo haciendo web scrapping, para luego ser ingresada en pandas. El único problema que se tuvo acá es que la información estadística acerca del juego es bastante escueta, es decir, como el juego tiene un formato de dados y más de acciones, su fuerte es más la estrategia de los movimientos, más que la estadística de los mismos.

Aquí ya obtuvimos las fuentes de datos para consultar y dar contexto al chatbot, ahora necesitamos un clasificador para entender a qué fuente consultar dependiendo la query o consulta. Para esto se plantearon dos modelos clasificadores, uno basado en regresión logística y otro basado en un LLM. El modelo de regresión logística se entrenó con muchas querys creadas por ChatGPT, que aun así desde mi punto de vista para la cantidad de información encontrada en las diferentes fuentes, esta no es suficiente para clasificar de manera correcta la consulta o query. Luego el modelo basado en LLM, aunque no tuviera mejores métricas en los datos de testeo con los cuales se testeó la regresión logística, captaba mejor la información que abarcaba cada fuente de datos, por lo que era más complicado engañar con prompts la misma, a diferencia del modelo de regresión logística, que para los prompts de testeo funcionaba bien, pero para prompts fuera del testeo, creados por mí basándose en la información de las bases tendía a fallar.

Una vez creado el clasificador, el siguiente paso es crear las funciones de búsqueda a las diferentes bases de datos, donde en la búsqueda a la base de grafos, la consulta a la misma tendría que ser dinámica, por lo que opté por usar el mismo LLM para que dado el formato de la base y el prompt, encontrar la mejor query a la base de grafos. Lo mismo con la base de datos tabular, en donde con la lista de columnas de la base y el prompt, se encontrará la mejor query para pandas. Para la búsqueda a la base de datos vectorial, se tenía que hacer una búsqueda híbrida, es decir, una búsqueda semántica utilizando el embedding del prompt y una búsqueda por palabras clave utilizando BM25, un algoritmo de búsqueda por palabras clave. También luego de utilizar ambas búsquedas era necesario utilizar un modelo de reranking para encontrar el mejor contexto para el prompt pasado.

Finalmente, para hablar con el chatbot experto en el juego de mesa, como toda mi información está en inglés, decidí que si el prompt era en español, me era necesario traducir el mismo, hacer toda la búsqueda y en el prompt al LLM agregar una regla de que sea necesario responder en el idioma original de la query.

Para el ejercicio 2 del trabajo, la consigna era crear un Agente inteligente basándose en herramientas creadas en el ejercicio 1. Para esto lo primero era elegir un modelo para descargar y probar, en la cátedra se recomendó el llama3.2 o el phi3, en mi caso utilicé el llama3.2. Luego había que crear funciones con un solo parámetro que sea la query o pregunta, ya que mis funciones tenían más parámetros, para que el agente pueda utilizarlas de manera sencilla. Se hostea el modelo en el puerto 8000, para poder utilizarlo. Finalmente, se crea el prompt para el modelo ReAct donde se pondrán todas las reglas o

instrucciones que tiene que seguir el modelo para buscar la información con sus diferentes herramientas y para responder la pregunta hecha.

Ejercicio 1:

Resumen

En este ejercicio se creará un chatbot experto en el juego de mesa Rajas of the Ganges, mediante un RAG y diferentes fuentes de conocimiento. Las fuentes de conocimiento van a ser 3 bases de datos, una vectorial, una de grafos y una tabular, donde la vectorial tendrá toda la información textual de reglas, reseñas, guías estratégicas y demás, la base de grafos tendrá información de diferentes relaciones del juego, como sus diseñadores, artistas, publicadores, idioma, premios, etc. Y la base de datos tabular tendrá información estadística o numérica del juego como rating, número de jugadores, tiempo de juego, edad mínima recomendada, jugadores totales, ranking, entre otros.
El contexto del RAG será obtenido mediante un clasificador del prompt que decidirá a que base de datos buscar la información, en el caso de buscar la información en la base de datos vectorial, se hará un rerankeo de los contextos encontrados por la búsqueda híbrida. Y las querys de la base de datos de grafos y tabular serán dinámicas.

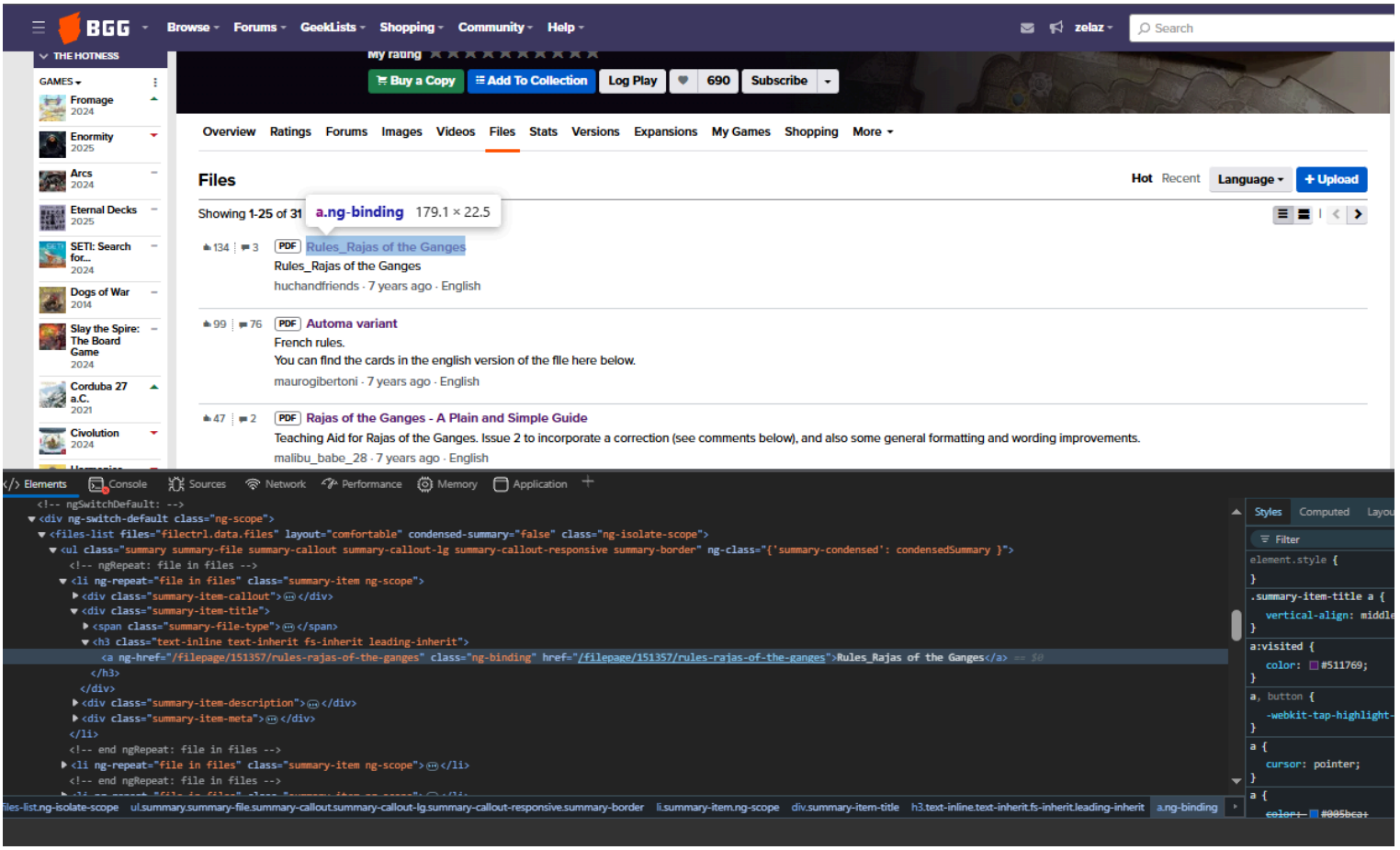
Metodología de resolución

Para este ejercicio lo primero era disponer de diferentes fuentes de datos, así como la base de datos vectorial, de grafos y tabular.

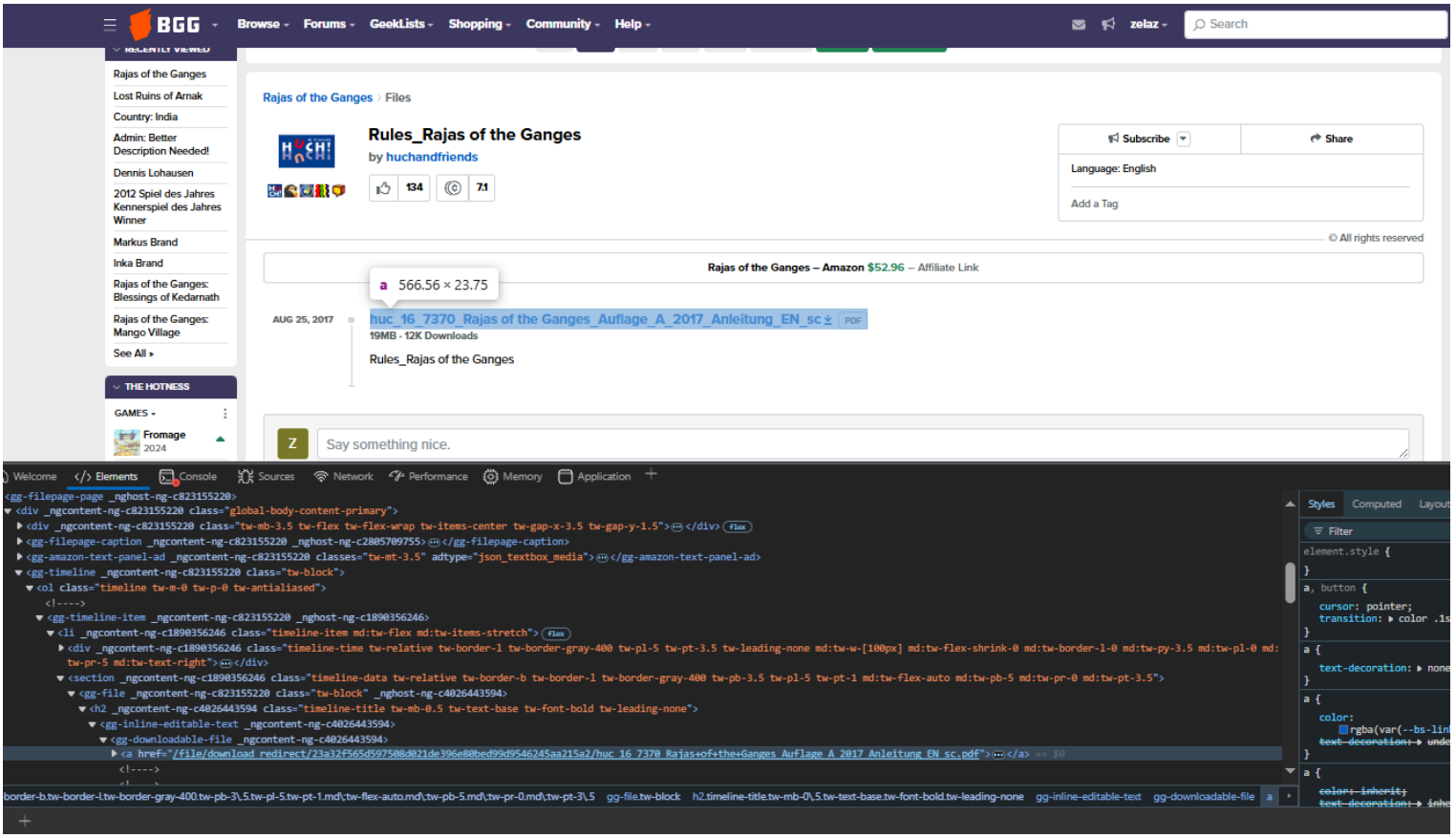
Base de datos vectorial

En esta base de datos, yo necesito guardar toda la información textual, entonces para eso se me ocurrió que las fuentes de datos iban a ser ayudas, reseñas, reglamentos, variantes en solitario y reglas rápidas, todo en inglés. Para conseguir estos datos, fue necesario ir a la sección de archivos en la página de boardgamegeek del juego, en mi caso pense diferentes maneras de descargarlos, uno siendo manualmente y subiéndolos a un Google Drive para que puedan ser utilizados automáticamente en el trabajo, y la otra sabiendo cuáles son los archivos, aplicar diferentes condiciones a Selenium para encontrar los links de los mismos, para ir a la página donde me deje descargar cada uno.

Ejemplo:



Encontrando ese link tendría que ingresar en esa página y una vez adentro encontrar el link de descarga y cliquearlo.



Entiendo que el propósito del trabajo en esta parte del ejercicio es utilizar web scrapping en la mayoría de los procesos de extracción de texto, pero vi innecesario utilizar Selenium para un trabajo que al fin y al cabo tendría que limitar a cuáles PDFs elijo yo manualmente, por lo tanto, decidí descargarlos manualmente y subirlos a drive.

Ahora lo siguiente era extraer el texto de diferentes reseñas de internet, como la de la página recomendada por la cátedra en el enunciado Reseña: Rajas of the Ganges | Misut Meeple en donde utilice BeautifulSoup para la extracción del texto en español, para después traducirlo con la librería deep_translator, separando el texto por “\n” o newlines, ya que el tamaño máximo de traducción es bastante reducido. También, utilice una reseña de otra página en inglés, donde de la misma manera que la reseña anterior, extraje toda la información y no fue necesaria traducirla.

Lo siguiente fue procesar todos estos documentos, que una vez fueron extraídos y en un caso traducido, que se llevaron a un Google Drive para no tener que repetir el proceso de extracción todo el tiempo. Para procesar estos, utilizo PyPDF2 para los pdfs y el propio Python para los TXT, ya que los archivos descargados de la página están en formato PDF y los archivos que extraje de las páginas de reseñas son guardados en formato TXT. Luego limpio todo este texto, buscando sacar caracteres que fueron mal reconocidos, espacios en blanco y diferentes errores encontrados en el texto. Después es necesario crear los chunks de texto, ya que para la búsqueda semántica en la base de datos vectorial va a ser más beneficioso si encuentro un párrafo de contexto que todo el documento. Para hacer estos chunks utilizo el SemanticChunker con el mismo modelo de embeddings que voy a utilizar para vectorizar las querys. Esto lo decidí porque el splitter recomendado en el enunciado del TP no era una mala idea, pero tenía muchas inconsistencias, también probé utilizar un splitter con kNN, pero este llevaba demasiados pasos y procesamiento para terminar teniendo chunks muy parecidos al primero, por lo que termine encontrando este splitter semántico, que mayormente cortaba en puntos y mantenía ideas en estos párrafos.

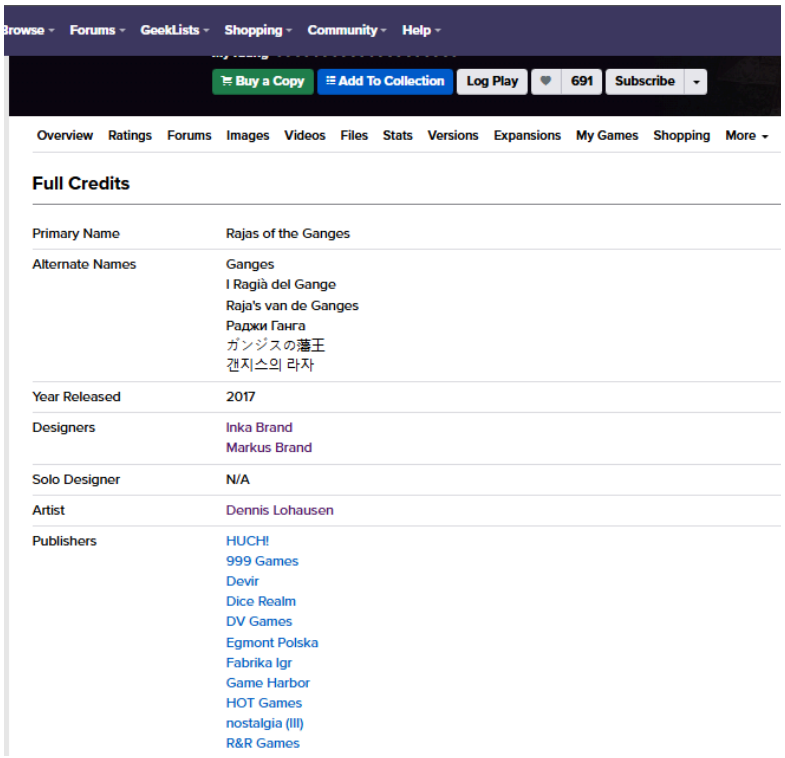
Ahora, una vez obtenidos los chunks, es necesario obtener el embedding de estos para guardarlos en la base de datos de chromaDB, obviamente utilizo el mismo modelo de embeddings del splitter que es “sentence-transformers/all-MiniLM-L12-v2”, el cual probé en el trabajo práctico anterior y me pareció el mejor.

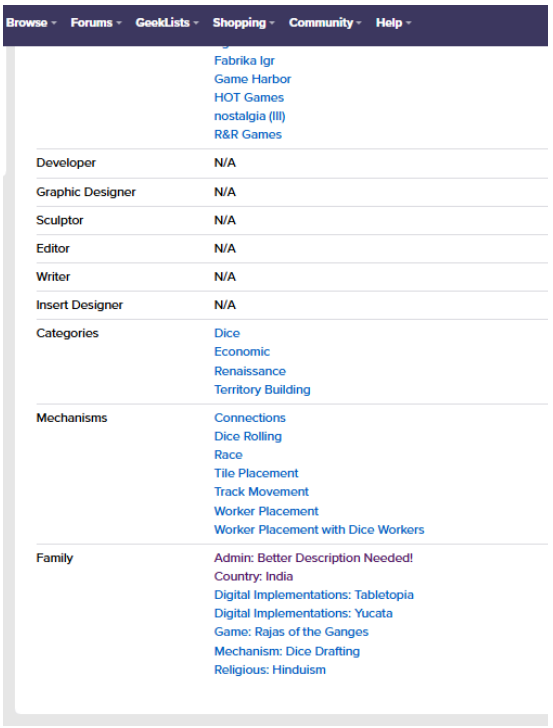
Para finalizar con la base de datos, se guardan todos los chunks junto con sus embeddings en la base.

Base de datos de grafos

Para esta base de datos la idea es tratar de encontrar datos que puedan visualizarse como relaciones, por ejemplo, los diseñadores, artistas, premios del juego, publicadores, etc.

Empiezo buscando información en la parte de créditos de la página, la cual tiene artistas, diseñadores, publicadores, categorías, mecánicas y familia. Mediante Selenium encuentro toda la información





Luego que obtengo toda esta información, también pense utilizar la relación de los diseñadores, sus premios y otros juegos en los que están involucrados como información extra, igual con el artista.

Utilizo RDF para hacer toda la base de datos de grafos, ya que me pareció que SPARQL tenía un formato de querys bastante parecido a SQL, fácil de utilizar y además teníamos mucha información en la teoría, ya que era una de las bases de datos de grafos explicada.

Mi idea de las relaciones diagramadas:

Con esta idea realizo todas las conexiones en RDF, en un momento también había pensado agregar el año de lanzamiento del juego, pero más adelante en la parte de clasificación entendí que es un valor numérico y para que mi clasificador pueda entender que ese dato específico no es una estadística o un valor numérico de la tabular, sino que un dato de relación, por lo que decidí dejarlo en la base de datos tabular.

Base de datos tabular

Para esta base de datos me encontré con que al ser un juego de mesa basado en acciones no se encuentran muchos datos estadísticos sobre el juego, por lo que la única información que tome para esta base de datos fue los datos encontrados en la sección de stats de la página boardgamegeeks, que son jugadores totales, ranking total, ranking estrategia, fans, dificultad, rating, año de lanzamiento, edad sugerida, tiempo de juego, número de jugadores y jugadores en el mes. Los datos que no se encuentran en esa sección están en el banner principal del juego. Decidí utilizar pandas DataFrame para realizar la base de datos tabular, y en caso de tener que exportarla, guardarla como CSV.

El único problema que veo con esta tabla es la poca cantidad de información que tiene, ya que no vi ningún dato numérico extra que pueda agregar a la misma.

bdd_tabular

	Number of Players	Play Time	Suggested Age	Release Year	Avg. Rating	Weight	Fans	Overall Rank	Strategy Rank	All Time Plays	This Month
0	2–4	45–75	12	2017	7.730	2.89 / 5	692	155	123	59.347	160

Clasificador de prompts

Una vez diseñamos y creamos las fuentes de datos, el siguiente paso es reconocer en el prompt de la persona qué información necesitamos, es decir, clasificar el prompt para saber a qué fuente de datos buscar esa información. Para esto había dos posibles implementaciones, una siendo un clasificador común con ML, en específico una regresión logística y, por otro lado, un clasificador basado en LLM. Para el clasificador con regresión logística era necesario tener un dataset para poder entrenar y testear el modelo, por lo que fue necesario utilizar ChatGPT para crear estas querys, luego se entrenó el modelo y se vieron sus métricas en testeo. Luego en el clasificador basado en LLM era necesario utilizar algún modelo LLM, por lo que, por temas de tamaño y capacidad computacional, se utilizó una API de huggingface para traer el modelo “Qwen/Qwen2.5-72B-Instruct” que funciona mucho mejor que otros probados como Zephyr o Mistral, además de ser una recomendación de la cátedra por su buen funcionamiento. Para clasificar las querys es necesario crear un prompt que le brinde suficiente información al LLM, ya que haciendo diferentes pruebas, con un prompt zero-shot, el LLM no llegaba a diferenciar entre que datos había en cada fuente, y el mejor clasificador fue con few-shot.

Después fue necesario comparar entre las métricas de los modelos en el conjunto de testeo del clasificador con regresión logística, ya que de otra forma no pueden ser comparados.

Modelo propio con regresión logística para la clasificación del prompt:

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

           0       0.96      1.00      0.98        24
           1       1.00      1.00      1.00        19
           2       1.00      0.94      0.97        17

    accuracy                           0.98        60
   macro avg       0.99      0.98      0.98        60
weighted avg       0.98      0.98      0.98        60
```

Modelo basado en LLM para la clasificación del prompt:

```
Reporte de clasificación de la LLM:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        24
           1       1.00      0.84      0.91        19
           2       0.85      1.00      0.92        17

    accuracy                           0.92        60
   macro avg       0.92      0.92      0.92        60
weighted avg       0.92      0.92      0.92        60
```

Luego vamos a comparar diferentes prompts que utilice para tratar de hacer fallar la clasificación y el desempeño de cada modelo con estos mismos.

Modelo basado en LLM:

- Valor esperado: reviews/rules

```
[42] query = 'what is the portuguese in the game?'

     print(classify_query(query=query, model="Qwen/Qwen2.5-72B-Instruct"))

     reviews/rules
```

- Valor esperado: statistics

```
[34] query = 'In what year was the game published?'

     print(classify_query(query=query, model="Qwen/Qwen2.5-72B-Instruct"))

     statistics
```

- Valor esperado: relations

```
     query = 'Who published this game?'

     print(classify_query(query=query, model="Qwen/Qwen2.5-72B-Instruct"))

     relations
```

- Valor esperado: reviews/rules

```
     query = 'How the karma influence the game?'

     print(classify_query(query=query, model="Qwen/Qwen2.5-72B-Instruct"))

     reviews/rules
```

- Valor esperado: reviews/rules

```
[52] query = 'what is the best strategy to start the game?'

     print(classify_query(query=query, model="Qwen/Qwen2.5-72B-Instruct"))

     reviews/rules
```

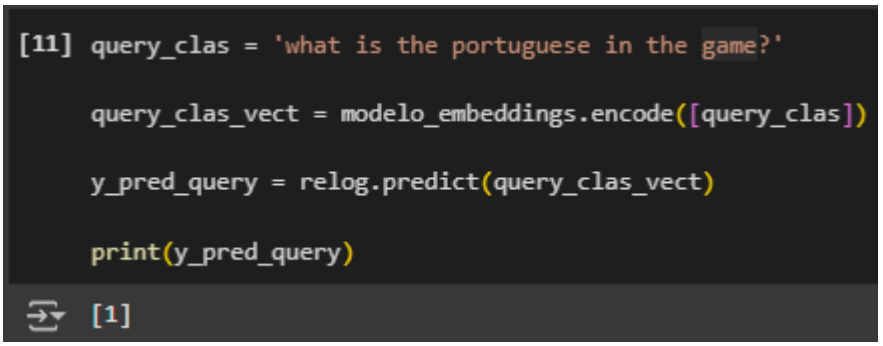
- Valor esperado: relations

```
[53] query = 'Where is the game from?'

     print(classify_query(query=query, model="Qwen/Qwen2.5-72B-Instruct"))

     relations
```

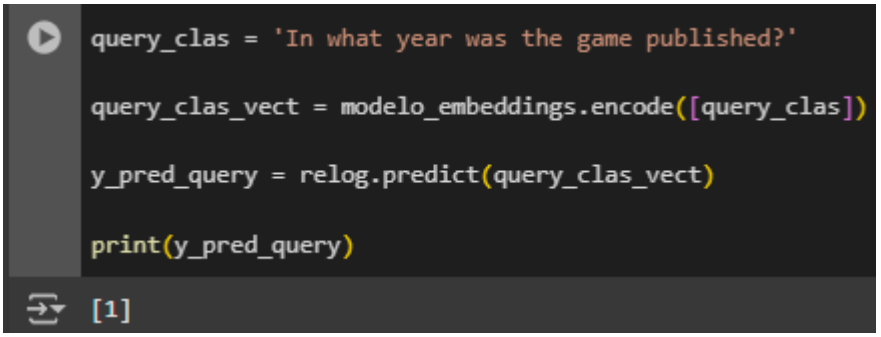
Modelo propio con regresión logística:

Aquí 0 es reviews/rules, 1 es relations y 2 es statistics

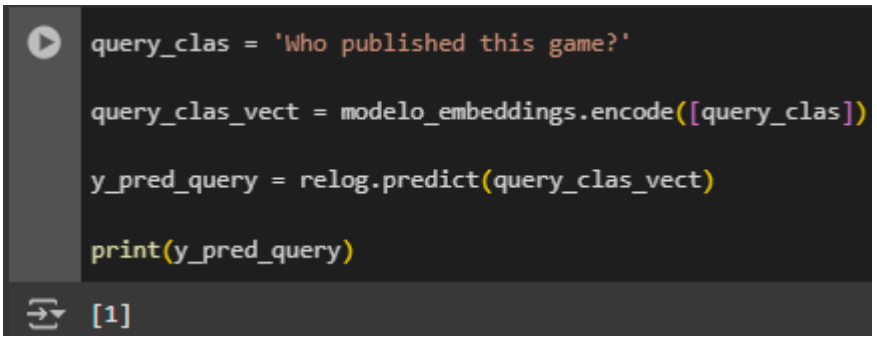
- Valor esperado: 0



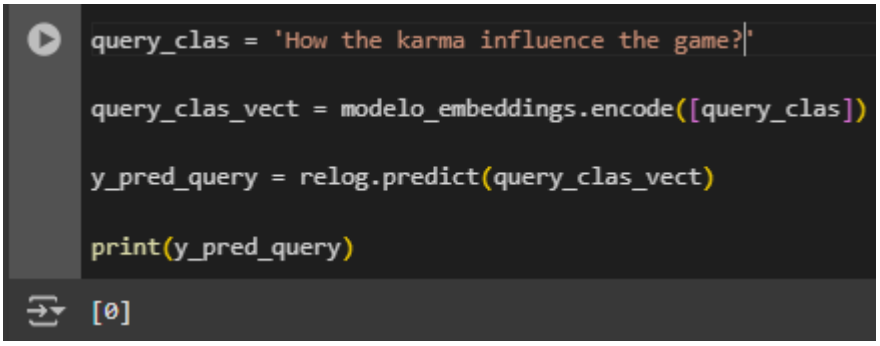
- Valor esperado: 2



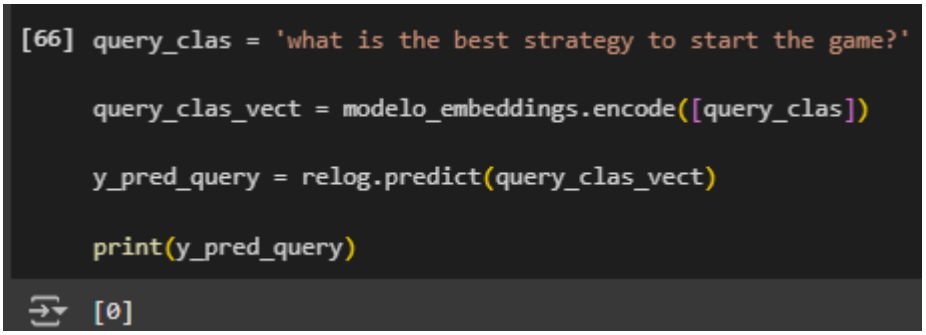
- Valor esperado: 1



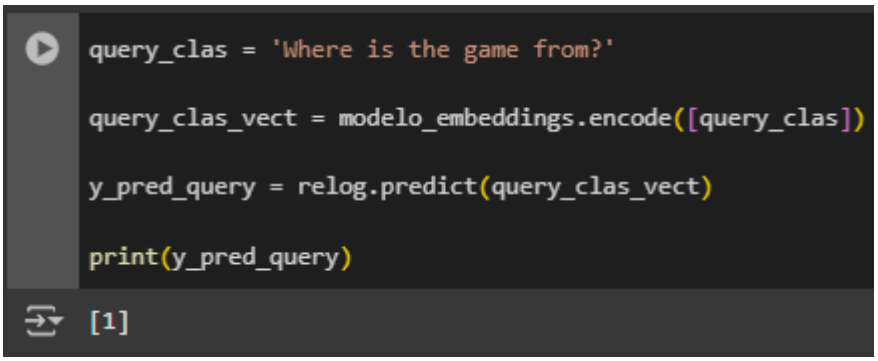
- Valor esperado: 0



- Valor esperado: 0



- Valor esperado: 1



Como se puede observar, el único que llega a fallar es el modelo propio, lo que no significa que sea el peor modelo, pero desde mi punto de vista al tener 100 prompts por tipo de prompt, no es un número considerable por la cantidad de información que se encuentra dentro de la información, entonces prompts que no conozca puede llegar a fallar, mientras que el modelo basado en LLM capaz no es el mejor en testeo, pero generalmente entiende mejor las categorías.

Chatbot experto con RAG

Llegado a este punto, nosotros ya tenemos todas las bases de datos hechas, el clasificador seleccionado y lo que faltaría son las querys dinámicas a la base tabular y de grafos, la búsqueda híbrida a la base vectorial, el reranker y el prompt de contexto al LLM final.

Primero vamos a ver la **búsqueda híbrida a la base de datos vectorial**, la cual por el lado de la búsqueda semántica, lo que se realiza es una query a la base de datos vectorial con el embedding del prompt de la persona y se traen los 5 mejores resultados, por el otro lado, se utiliza el algoritmo de búsqueda por palabras clave BM25, para hacer una búsqueda por palabras clave, donde la query va a pasar ciertas limpiezas y la tokenización, para luego ser utilizada por el algoritmo y encontrar las 5 mejores respuestas.

Luego de hacer esta búsqueda híbrida es necesario un reranking de las respuestas encontradas por ambos métodos, este proceso consta de un modelo cross-encoder el cual va a recibir las 10 respuestas encontradas por los métodos y va a realizar un ranking de qué respuesta responde mejor al prompt, el modelo utilizado fue el “BAAI/bge-reranker-v2-m3”, que fue recomendado por la cátedra.

Lo segundo es hacer las querys dinámicas a la base de grafos y tabular, para esto se me ocurrió una sola forma de hacerlo, que consta de utilizar un LLM y junto con el formato de la base de datos, obtener una query para la misma. En el caso de la base de datos tabular con pasar las columnas del dataframe es suficiente.

Query dinámica en la base de datos de grafos

En mi caso, utilizo el siguiente mensaje de contexto para el LLM en la búsqueda en la base de datos de grafos.

```
messages = [
  {
    "role": "user",
    "content": f""" I have a RDF graph database with the following format: {g.serialize(format="turtle")}.
  I want you to write a query in RDF to answer the following question:
  '{query}'.

  Return me only the query in RDF, without explanation or additional text.
  Make sure you only return clean, functional code, without characters like \\n at the beginning or end.
  """
  }
]
```

En un principio, este prompt no tenía problemas con las preguntas de prueba, pero en un momento, decidí preguntarle “who are the designers of the game” y me devolvió algo que no era ni una query ni nada, era un mensaje totalmente sin sentido, por lo que pense en ajustar un poco el prompt del LLM, ya que si no se rompería con cualquier pregunta sencilla de esta índole.

```
messages = [
  {
    "role":"system",
    "content":f""" You are an assistant that will be responding only RDF querys.
    """
  },
  {
    "role": "user",
    "content": f""" I have a RDF graph database with the following format: {g.serialize(format="turtle")}. Always follow the database format.
  I want you to write a query in RDF to answer the following question:
  '{query}'.

  Return me only the query in RDF, without explanation or additional text.
  Make sure you only return clean, functional code, without characters like \\n at the beginning or end. Be sure to not add characters or words before or after the query.
  """
  }
]
```

Cambiando el prompt, conseguí tener buenos resultados en la pregunta anterior, obviamente no es el prompt perfecto, pero se va ajustando dependiendo de las respuestas.

Query dinámica en la base de datos tabular

Para la base de datos tabular utilizo el siguiente mensaje de contexto.

```
messages = [
  {
    "role": "user",
    "content": f""" I have a Pandas DataFrame with the following columns: {list(bdd_tabular.columns)}.
  I want you to write a query in pandas to answer the following question:
  '{query}'.

  Return me only the query in pandas, without explanation or additional text.
  Make sure you only return clean, functional code, without characters like \\n at the beginning or end.
  """
  }
]
```

En esta query dinámica tuve el problema de que no reconoce el nombre de la base de datos, por lo que siempre toma como nombre de la base de datos como df y tenía que hacer un replace en la string de la query para que se pueda buscar en pandas. Teniendo en cuenta esto, intenté agregarle contexto informándole cuál era el nombre de la base de datos.

```
messages = [
  {
    "role": "user",
    "content": f""" I have a Pandas DataFrame with the following columns: {list(bdd_tabular.columns)}.
  I want you to write a query in pandas to answer the following question:
  '{query}'.

  Return me only the query in pandas, without explanation or additional text. The database name is: bdd_tabular. Be sure to use this name in the query.
  Make sure you only return clean, functional code, without characters like \\n at the beginning or end.
  """
  }
]
```

Con este prompt el LLM llega a entender el nombre de la base de datos, ya que probando con diferentes prompts, la query tenía ya de por sí el nombre de la base que le pase. Ahora el problema era que intentaba utilizar funciones de pandas para encontrar mínimos, máximos y otros valores, pero yo al tener un dataframe de una sola fila, estas querys terminan rompiendo el código por no poder realizarse, además, mi dataframe en algunas columnas cuenta con valores de string, por lo que estas funciones fallarían por ese lado, a lo que intente darle contexto de que tan grande era el dataframe y que no le agregue funciones extra, sino la búsqueda sencilla. También en algunas ocasiones me agregaba “```python” al principio de la query de pandas o “```” al final de la query, por lo que le tuve que indicar que no agregue strings al principio o final de la query de pandas. Terminando de esta manera el prompt.

```
messages = [
  {
    "role": "user",
    "content": f""" I have a Pandas DataFrame with the following columns: {list(bdd_tabular.columns)}. The DataFrame counts with only one row, so don't add more commands to the query than the column name.
  I want you to write a query in pandas to answer the following question:
  '{query}'.

  Return me only the query in pandas, without explanation or additional text. The database name is: bdd_tabular. Be sure to use this name in the query.
  Make sure you only return clean, functional code, without characters like \\n at the beginning or end. Be sure to not add characters or words before or after the query.
  """
  }
]
```

Chatbot LLM

Por último, una vez definidos todos los métodos de búsqueda para cada base de datos, una vez se hace un prompt, se chequea el idioma del prompt, en caso de estar en español se traduce y se guarda el idioma, para obligar al LLM a responder en español y en caso de ser inglés procede todo normal. Luego se clasifica el mismo en inglés obligatoriamente, se busca la información dependiendo del tipo de clasificación de la query, en donde primero se busca por el tipo de clasificación, pero en el hipotético caso de que no se encuentre la información en la base de datos de grafos o en la base de datos tabular, se hace la búsqueda en la base de datos vectorial, por el simple hecho de intentar devolver alguna información.

El mensaje de contexto al LLM es.

```
messages = [
  {
    "role": "user",
    "content": f""" Answer the question. The response needs to be in {lang}, clear and without extra information. with that context: {context}

    Question: {query}
    Answer:

    """
  }
]
```

Pero con este prompt, sigue habiendo contextos a los que el LLM no le encuentra sentido, por ejemplo, al preguntarle sobre rajas, es decir, el juego, en muchas ocasiones me devolvía como nombre del juego Rummikub, juego de mesa que existe, pero no es el objetivo de mi chatbot, por lo que cambie el prompt y le añadí un mensaje de sistema para que entienda que es experto en rajas of the ganges, que todas las preguntas acerca de un juego van a ser de rajas, etc.

```
messages = [
  {
    "role": "system",
    "content":f"""
    You are an assistant that will be expert in the boardgame Rajas of the Ganges, any question about an game would be about the Rajas.
    """
  },
  {
    "role": "user",
    "content": f""" Answer the question. The answer needs to be in {lang}, clear and without extra information. with that context: {context}

    Question: {query}
    Answer:

    """
  }
]
```

Aquí ya el LLM parecía estar respondiendo la mayoría de las preguntas bien, quedan algunas sutilezas como que la dificultad del juego, si se pregunta de esta manera: “cuál es la dificultad del juego” el clasificador lo entiende como un valor de la base vectorial, pero este en verdad es un valor numérico de la base tabular, por lo que entiendo que muchas preguntas pueden ser respondidas con el prompt ideal, como cuando vimos en la teoría que muchas veces la respuesta del chatbot depende de la ingeniería del prompt, en que tan bien uno puede expresar la pregunta en el formato que entiende el chatbot. Por ejemplo, la pregunta de la dificultad es necesario realizarla de esta manera para que funcione:

```
Ingrese su imput: in the scale of 1 to 5 what is the difficulty of the game?
bdd_tabular['Difficulty']
2.89 / 5
```

Vemos la pregunta, también un print de la query de pandas y el valor devuelto, todavía vemos que le cuesta en ciertas ocasiones responder en formato de respuesta. Pero la respuesta a la pregunta es esa, por lo que tampoco quiero seguir cargando el prompt del LLM para que no empiece a delirar o hacer diferentes extensiones de respuesta que estén mal.

Finalmente, entiendo que se puede seguir haciendo una cantidad inmensa de modificaciones en los prompts de todas las partes, sea del clasificador, del LLM que obtiene el contexto, el LLM final y también se pueden hacer cambios de modelos, probando alguno de otro desarrollador, la temperatura de la respuesta, entre muchas otras cosas. También

depende de como uno pregunta, porque aunque uno tenga el prompt bastante bien armado, una pregunta sin sentido sigue haciendo fallar el entendimiento del LLM.

Por ejemplo:

Si yo pregunto: how the portuguese works in the game?
El LLM responde: The Portuguese action in Rajas of the Ganges allows players to move their worker along the river track to gain certain bonuses. These bonuses can include additional workers, resources, or other benefits. The key aspect of the Portuguese action is that it allows players to advance on the river track using lower-value dice, which can be particularly useful for obtaining new workers or other valuable bonuses. However, players should be cautious not to focus solely on this action, as it can cause them to miss out on other important bonuses along the way.

Esto es una respuesta solida y que responde bien la preguntas, el tema es cuando la pregunta no es lo suficiente explicita.

Si yo pregunto: what is the portuguese in the game?
El LLM responde: In the context of the game, the Portuguese are not a specific game element. However, the term "Grão Mogol," which is Portuguese for "Great Mogul," is mentioned in the historical context provided. This reflects the Portuguese influence in the Indian Ocean during the time of the Mogul Empire.

El LLM parece delirar sobre esta pregunta y trata de encontrar algun sentido con el idioma, lo que esta mal. Pero no me parece que sea problema del modelo en sí, sino sus limitaciones en base a lo poco representativa que es la pregunta.

Luego se pueden hacer preguntas como:

```
Ingrese su pregunta de Rajas of the Ganges: quienes son los diseñadores?
Los diseñadores son Inka Brand y Markus Brand.
```

```
Ingrese su pregunta de Rajas of the Ganges: quien diseño el juego?
Los diseñadores del juego son Inka Brand y Markus Brand.
```

En un principio el LLM entendía la segunda pregunta como que habia un solo diseñador y devolvia una sola, pero vemos que al darle un prompt más complejo, este entiende que si se devuelve dos valores es porque son más de un diseñador, y en este caso esta pequeña diferencia en la pregunta, no hace fallar el modelo.

```
Ingrese su pregunta de Rajas of the Ganges: en que año fue lanzado el juego?
2017
```

En esta pregunta anteriormente tuve el problema de que el LLM me cambiaba el nombre de la columna de la base tabular y daba problemas, pero con los cambios hechos anteriormente, se observa que devuelve el valor. Eso sí, vemos que la respuesta es corta y sin texto, lo que quisiera tratar de cambiar, pero al darle la instrucción de responder de manera larga, podría hacer que añada información en otras preguntas, lo cual no es algo que quiero que suceda.

Todas estas pruebas anteriormente mostradas fueron hechas con el modelo Qwen/Qwen2.5-72B-Instruct, el cual tiene una cantidad de parámetros bastante grande, pero también pense en hacer algunas pruebas con algunos modelos más chicos como el microsoft/Phi-3.5-mini-instruct, pero en general la mayoría de las preguntas al tener el contexto encontrado por nosotros contesta de manera correcta, en el apartado de querys dinámicas parecía tener un buen rendimiento, ya que con preguntas sencillas no tenía ningún error. Luego intenté con mistralai/Mistral-7B-Instruct-v0.3, que en algunas preguntas no sabía hasta donde llegaba la información de la pregunta, y terminaba agregando información del contexto que capaz no era necesaria en la respuesta. En el apartado de querys dinámicas este modelo flaqueaba, ya que no entendía que eran bases de datos en pandas, o como hacer las mismas.

En general, Qwen/Qwen2.5-72B-Instruct es superior en la mayoría de los aspectos, obviamente siendo mucho más pesado computacionalmente y con 72b de parámetros.

Conclusiones

Mi conclusión en este punto fue que en general me pareció un ejercicio interesante, el cual capaz no se tuvo el tiempo suficiente para ajustar de manera fina la mayoría de los puntos a realizar, desde mi punto de vista hubiera estado bueno empezar el trabajo al mismo tiempo que dábamos la unidad 5. Aun así, el chatbot desde mi punto de vista funciona bastante bien, responde la mayoría de las preguntas, me imagino que entre más información uno quiere abordar con este RAG más complejo se vuelve todo este trabajo, yo creo que dentro de todo no sé si entendí mal el punto, pero no encontré una gran cantidad de información, sin que se repita por el idioma o etc. En la parte inicial de la extracción de datos, yo me imagino que mayormente es necesario el web scrapping, pero quedo poco claro, ya que al haber pdfs que se podían descargar fácilmente, era más sencillo descargarlos manualmente y subirlos a Google Drive para su uso, en las páginas de reseñas si era necesario. Luego no entendí la parte de si era necesario del todo buscar información en Wikipedia para este trabajo, ya que lo único que aportaba esto era información geográfica, o histórica acerca de la ambientación del juego y no estaba seguro de que si el chatbot tenía que ser experto en el juego de mesa, esto implicaba también sobre su contexto histórico. Por último, la parte de ajuste de los prompts a los LLM me pareció la parte más interesante, ya que modificando este mismo en un solo modelo, uno obtiene una infinidad de respuestas diferentes. Si también pensamos en cambiar el modelo del LLM y con la cantidad disponible en HuggingFace, uno puede ajustar y probar una cantidad de prompts y modelos que no da el tiempo.

Librerías y modelos

Librerías

Librerías de web scrapping:

- **Selenium**: Selenium
- **BeautifulSoup:** beautifulsoup4 · PyPI
- **requests:** requests · PyPI

Librerías de bases de datos:

- **Pandas:** pandas - Python Data Analysis Library
- **chromadb:** Chroma
- **RDFLib:** rdflib 7.1.1 — rdflib 7.1.1 documentation
- **json:** json — JSON encoder and decoder — documentación de Python - 3.13.1
- **urllib:** urllib — URL handling modules — documentación de Python - 3.13.1

Librerías de extracción y modificación de texto:

- **PyPDF2:** PyPDF2 · PyPI
- **deep_translator:** deep-translator · PyPI
- **langdetect:** langdetect · PyPI
- **unidecode:** Unidecode · PyPI
- **nlkt:** NLTK :: Natural Language Toolkit
- **re:** re — Regular expression operations — documentación de Python - 3.13.1
- **spacy:** spaCy · Industrial-strength Natural Language Processing in Python

Librerías de propósitos generales:

- **Time:** time — Time access and conversions — Python 3.13.1 documentation
- **google.colab.files:** External data: Local Files, Drive, Sheets, and Cloud Storage - Colab
- **os:** os — Miscellaneous operating system interfaces — Python 3.13.1 documentation
- **gdown:** gdown · PyPI

Librerías de ML:

- **NumPy:** NumPy -
- **sklearn:** scikit-learn: machine learning in Python — scikit-learn 1.6.0 documentation

Librerías de RAG y LLM:

- **FlagEmbedding:** FlagEmbedding · PyPI
- **rank_bm25:** rank-bm25 · PyPI
- **sentence_transformers:** SentenceTransformers Documentation — Sentence Transformers documentation
- **langchain:** Introduction | 🦜🔗 LangChain
- **huggingFace_hub:** Hugging Face Hub documentation

Modelos

- **Embeddings de querys:** sentence-transformers/all-MiniLM-L12-v2 · Hugging Face
- **Embeddings de chromadb:** sentence-transformers/all-MiniLM-L12-v2 · Hugging Face
- **Reranker:** BAAI/bge-reranker-v2-m3 · Hugging Face
- **Clasificador de querys:** Qwen/Qwen2.5-72B-Instruct · Hugging Face **(API)**
- **Querys dinámicas de base de grafos:** Qwen/Qwen2.5-72B-Instruct · Hugging Face **(API)**
- **Querys dinámicas de base de datos tabular:** Qwen/Qwen2.5-72B-Instruct · Hugging Face **(API)**
- **Chatbot final:** Qwen/Qwen2.5-72B-Instruct · Hugging Face **(API)**

Ejercicio 2:

Resumen

En este ejercicio se creará un agente inteligente experto en el juego de mesa Rajas of the Ganges, mediante un modelo de ReAct y diferentes fuentes de conocimiento. Las fuentes de conocimiento van a ser 3 bases de datos, una vectorial, una de grafos y una tabular, donde la vectorial tendrá toda la información textual de reglas, reseñas, guías estratégicas y demás, la base de grafos tendrá información de diferentes relaciones del juego, como sus diseñadores, artistas, publicadores, idioma, premios, etc. Y la base de datos tabular tendrá información estadística o numérica del juego como rating, número de jugadores, tiempo de juego, edad mínima recomendada, jugadores totales, ranking, entre otros.

Metodología de resolución

En este ejercicio, no tendremos una gran cantidad de pasos a realizar, ya que sus fuentes de conocimientos y funciones que dejan buscar en estas fuentes ya están realizadas, por lo que lo único que queda por realizar es el agente inteligente.

Funciones de búsqueda

Para las funciones de búsqueda del agente inteligente, aprovecharemos las funciones antes hechas en el primer ejercicio, haciendo un pequeño cambio y hardcodeando el modelo LLM para las querys dinámicas y no mucho más.

Agente

Para el agente lo primero va a ser realizar todo el prompt de ReAct, en el cual se le dará todo el contexto, forma de pensar, herramientas disponibles, reglas, formatos de respuesta, formatos de parámetros, entre otros. Para eso lo primero que se me ocurrió fue que al tener toda la base de datos en inglés, todas estas instrucciones tendría que ser en inglés. El modelo de agente a utilizar es **llama3.2:3b**

El primer prompt que fue probado tenía un formato parecido a esto:

You are an assistant that can respond in Spanish or English depending on the query. You need to follow restrictively the next rules:

Thought: here i think what i need
Action: tool_name (one of {tool_names}) if using a tool.
Action Input: the input to the tool, in string format.

Available tools:
{buscar_en_grafos}: Action Input: "search text"
{buscar_en_tabular}: Action Input: "search text"
{buscar_en_vectorial}: Action Input: "search text"

Tools themes:
- **buscar_en_grafos: DESIGNERS, ARTISTS, GAME LANGUAGES, PUBLISHERS, OTHER LANGUAGES, AWARDS.**
- **buscar_en_tabular: STATISTICS IN GENERAL, RELEASE YEAR, MINIMUM AGE RECOMMENDED, TIME PLAY, AMOUNT OF PLAYERS.**
- **buscar_en_vectorial: GENERAL RULES, AUTOMATA RULES, NAVARATNA RULES, GAME OBJECTIVE, COMPETITOR STRATEGY, BEGINNER STRATEGY, PORTUGUESE CHARACTER STRATEGY**

Interaction examples:
Query 1:
- "How the portuguese works in the game?"
- Thought: I need to search about game rules.
- Action: buscar_en_vectorial
- Action Input: "What is the portuguese in the game?"

Query 2:
- "Who did the artwork for this game?"
- Thought: I need to search about artists.
- Action: buscar_en_grafos
- Action Input: "Who did the artwork for this game?"

Query 3:
- "What is the minimum recommended age to play the game?"
- Thought: I need to search about game rules.
- Action: buscar_en_tabular
- Action Input: "What is the minimum recommended age to play the game?"

Observation: [Result of the tool]
... [Repeat the process if needed]

Final Answer: The final answer, combining all the information

Instructions for each query:
1. Analyze what information you need
2. Use the all the available tools one per one with the correct format.
3. Combine all the results in one final answer.

Additional rules:
- Never modify the user query: Every time you receive a new query, forget the previous one. The new query must be treated independently and without previous influences.
- Prioritize the tools by the relation theme of the query. Secure yourself by selecting the best tool that adjust to the query with the proportioned themes.
- Never response with information that does not provide directly from the tools.
- The output format must be strict and follow the example proportioned.
- You MUST obey the function signature of each tool. Do NOT pass in no arguments if the function expects arguments.

IMPORTANT:
- Always follow the exact format of the examples for each tool.
- If the query requires information of multiples tools, you MUST use ALL the tolls needed to obtain a FULL answer
- Don't skip any relevant tool, although you have part of the answer.
- Never use the output of a tool to use another tool. Only use the user query to use the tools.

Lo que tenía este prompt es que mayormente el modelo deliraba con los parámetros de las herramientas, aunque encontrara la respuesta, quedaba en loop infinito de búsqueda y mayormente no respondía nada.

En este punto decidí ir a la documentación de llama_index y encontré un formato de prompt bastante interesante, adopte algunos cambios basándome en el formato default encontrado en esa página.

Este formato, entre otras cosas, no definía ejemplos de querys, no le daba un solo chain of thought, sino que también daba diferentes chain of thought en caso de encontrar una respuesta, tenía menos reglas, pero más concisas y no le daba instrucciones como yo le definía. También un problema que tenía era que detectaba el idioma de la query en español, aunque sea completamente en inglés, por lo que mi única idea de cuál podía ser el problema era las funciones en español, por lo que las cambie a inglés.

El prompt combinado de mi prompt inicial más el de la documentación quedó de esta manera:

You are an assistant that can respond in Spanish or English depending on the query. You need to follow restrictively the next rules:

Tools
You have access to a wide variety of tools. You are responsible for using the tools in any sequence you deem appropriate to complete the task at hand. This may require breaking the task into subtasks and using different tools to complete each subtask.

Available tools:
{search_in_graph}: Action Input: "search text"
{search_in_tabular}: Action Input: "search text"
{search_in_vectorial}: Action Input: "search text"

Tools themes:
- **search_in_graph: DESIGNERS, ARTISTS, GAME LANGUAGES, PUBLISHERS, OTHER LANGUAGES, AWARDS.**
- **search_in_tabular: STATISTICS IN GENERAL, RELEASE YEAR, MINIMUM AGE RECOMMENDED, TIME PLAY, AMOUNT OF PLAYERS.**
- **search_in_vectorial: GENERAL RULES, AUTOMATAS RULES, NAVARATNA RULES, GAME OBJECTIVE, COMPETITOR STRATEGY, BEGINNER STRATEGY, PORTUGUESE CHARACTER STRATEGY**

Tools Parameters:
- Always use only the query text to call the tools.
- Never use another information to be part of the parameters.
- The tools only accept strings in the parameters.

Output Format
To answer the question, please use the following format.

```
**Thought: I need to use a tool to help me answer the question**
**Action: tool_name (one of {tool_names}) if using a tool.**
**Action Input: the input to the tool, in string format.**
```

Please ALWAYS start with a Thought.

Please use a valid string format for the Action Input. Do NOT do this {{'input': 'hello world', 'num_beams': 5}}.

If this format is used, the user will respond in the following format:

```
Observation: tool response
```

**You should keep repeating the above format until you have enough information to answer the question without using any more tools. At that point, you MUST respond
in the one of the following two formats:**

```
Thought: I can answer without using any more tools.
Answer: [your answer here]
```

```
Thought: I cannot answer the question with the provided tools.
Answer: Sorry, I cannot answer your query.
```

Additional rules:
- Never modify the user query: Every time you receive a new query, forget the previous one. The new query must be treated independently and without previous influences.
- Prioritize the tools by the relation theme of the query. Secure yourself by selecting the best tool that adjust to the query with the proportioned themes.
- Never response with information that does not provide directly from the tools.
- The output format must be strict and follow the example proportioned.
- You MUST obey the function signature of each tool. Do NOT pass in no arguments if the function expects arguments.

IMPORTANT:
- Always follow the exact format of the examples for each tool.
- If the query requires information of multiples tools, you MUST use ALL the tolls needed to obtain a FULL answer
- Don't skip any relevant tool, although you have part of the answer.
- Never use the output of any tool to use another tool. Only use the user query to use the tools.

Current Conversation
Below is the current conversation, consisting of interleaving human and assistant messages.

Utilizando este prompt, pude hacer funcionar muchas más preguntas, algunas siguen sin poder ser resueltas, pero tengo la hipótesis que en este agente para que funcione bien todo el chain of thought, es decir, que el modelo entienda que un contexto es la respuesta de una pregunta, esta pregunta tiene que ser más específica a diferencia del ejercicio anterior del RAG, donde al nosotros encontrar ese contexto y "obligar" al LLM a responder con el mismo, uno se ahorraba todo ese razonamiento. En este caso, si una pregunta carece de sentido o no es lo suficientemente especifica, el agente no va a saber entender que el contexto encontrado es la respuesta a la pregunta. Por lo que uno tiene que ser cuidadoso con las preguntas al agente y tratar de que sean más descriptivas y asociadas al contexto que piensa que se le va a asociar.

Un ejemplo de este problema fue cuando le pregunte "how the portuguese works in the game?", y el agente desde mi punto de vista entendió que el portugués podría ser una persona y al obtener el contexto de la base de datos vectorial, me respondió que no tenía la información necesaria para responder la pregunta, pero al reformular la misma de esta manera "how the portuguese mechanic works in the game?", encontraba el mismo contexto y terminaba dando una respuesta, ya que le encontraba sentido entre ambas cosas.

Luego tuve el problema de que al preguntarle sobre el año de lanzamiento del juego, encontró el contexto, buscó en otras fuentes, ya que el año solo capaz era un dato pequeño y tiene la orden de buscar en varias fuentes hasta tener una respuesta sólida. El tema fue que encontró el año, lo utilizó como input de otra herramienta, lo que me pareció otro error a corregir, pero también no supo que hablaba de rajas of the ganges, lo que al buscar en la base vectorial, encontró la palabra Master Builder y la utilizó como nombre de juego. Para esto pensé en cambiar el prompt, agregándole una línea inicial de que es experto en rajas of the ganges y que la mayoría de las preguntas de un juego van a ser basadas en ese juego.

El cambio fue en la parte inicial, donde le ingrese esta regla luego de explicarle que es un asistente de IA:

You are an expert in the boardgame Rajas of the Ganges, all the questions about a game would be about the Rajas of the Ganges.

Con este nuevo prompt, probé la query anterior y obtuve la respuesta del año. Luego probé las querys que venía utilizando para probar las diferentes fuentes de información, así como "how the portuguese mechanic works in the game?", "who are the designers of the game?", y ambas tuvieron el mismo resultado, por lo que fue un cambio positivo, aun así al probar la pregunta del año de lanzamiento del juego, obtuve diferentes respuestas, en un caso

parece delirar con los parámetros y no obtiene una respuesta, pero en otros casos ingresa bien los parámetros y obtiene bien la respuesta.

Probando diferentes querys, observe que el funcionamiento del agente era inestable, mi hipótesis es que al tener las funciones con parámetros en español este muchas veces identificaba mal como era el formato de los parámetros por tener idiomas distintos, y terminaba utilizando cualquier nombre de parámetro. Decidí cambiar todas las funciones y dejar sus parámetros en inglés, luego probé con las mismas preguntas y en efecto los problemas que tenía con como se pasan los parámetros y las fluctuaciones en el funcionamiento mejoraron.

Luego probé con una query que tendría que extraer información de la base de grafos, en específico los publishers del juego, los cuales son una lista larga de nombres. En un principio tiene problemas para hacer el parse de la respuesta, pero después de varios intentos, entiende como extraer la información y devuelve una respuesta correcta.

También comento que algo que fue de ayuda en el agente fue utilizar el parámetro **context=You are an useful assistant that responds in spanish or english depending on the query. Remember the outputs of all the tools to use them in the final answer. You are an expert in the boardgame Rajas of the Ganges, all the questions about a game would be about the Rajas of the Ganges.**
Que es de ReActAgent.from_tools, el cual me solucionó algunos problemas que tenía en querys en donde el agente no entendía el contexto de qué juego se le hablaba y terminaba delirando o respondiendo mal.

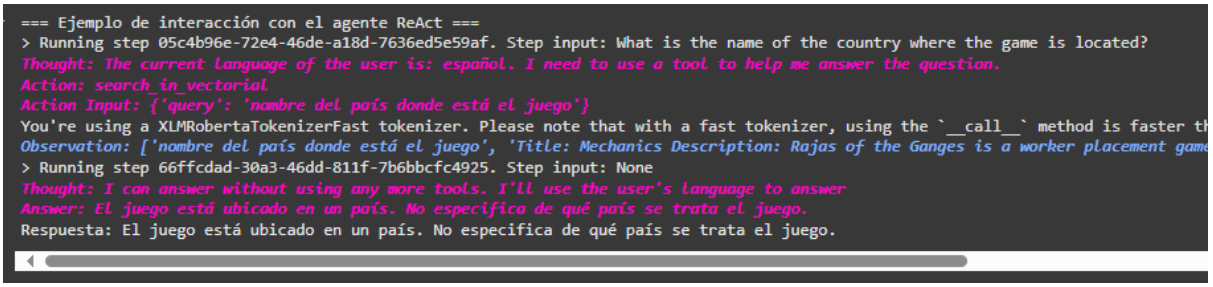
Otro problema que tuve, por lo que decidí no utilizar esa herramienta, es wikipedia, ya que lo intenté implementar en el agente y por más que tratara de cambiar el prompt, poniendo reglas o instrucciones para que lo use una vez o como último recurso, no tuvo caso, el modelo siempre terminaba llamando a wikipedia, y al ser el agente el que ingresa el parámetro, este ingresaba algo no tan exacto y wikipedia respondía información fuera de contexto, la cual hacía que el agente funcionara peor, ya que trata de llamarlo de nuevo y así. Mi idea era que una vez buscara en las herramientas principales, en caso de no encontrar la información necesaria buscará en wikipedia, pero terminaba buscando cuando quería. Además, justamente como comentaba anteriormente la query con la que ingresaba a wikipedia el agente mayormente no era el correcto y esto arrastraba un error inmenso, que ya una vez hacía la llamada a wikipedia, era casi imposible que encontrara una respuesta más adelante.

Pruebas con otros modelos:

Luego de encontrar un prompt decente para el modelo llama3.2 decidí probar otros modelos, con el mismo prompt y estos fueron los resultados:

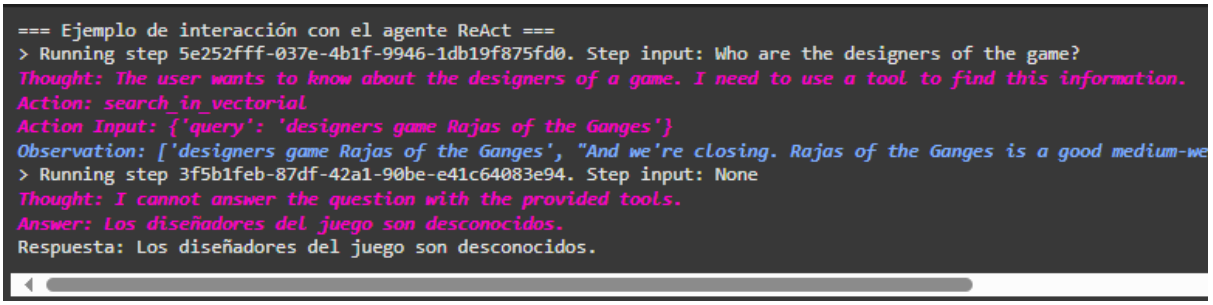
qwen2.5:3b

What is the name of the country where the game is located?



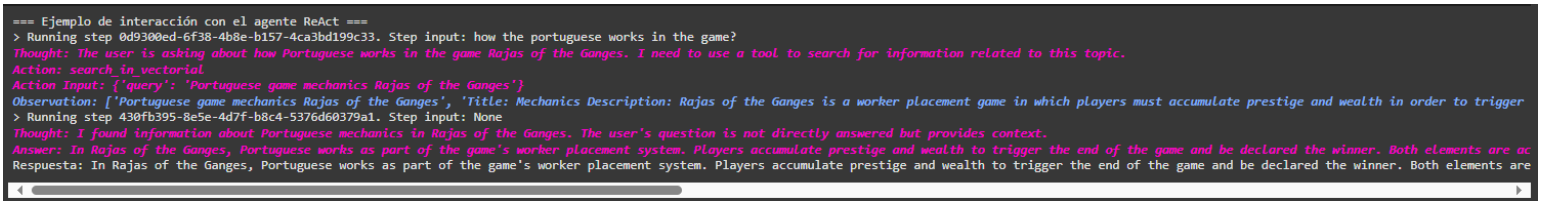
En este ejemplo, vemos que lo único que hace es ir a buscar en la base de datos vectorial y razonar que el juego está ubicado en algún país, pero obviamente la información no esta en esa base, sino en la base de grafos.

Who are the designers of the game?



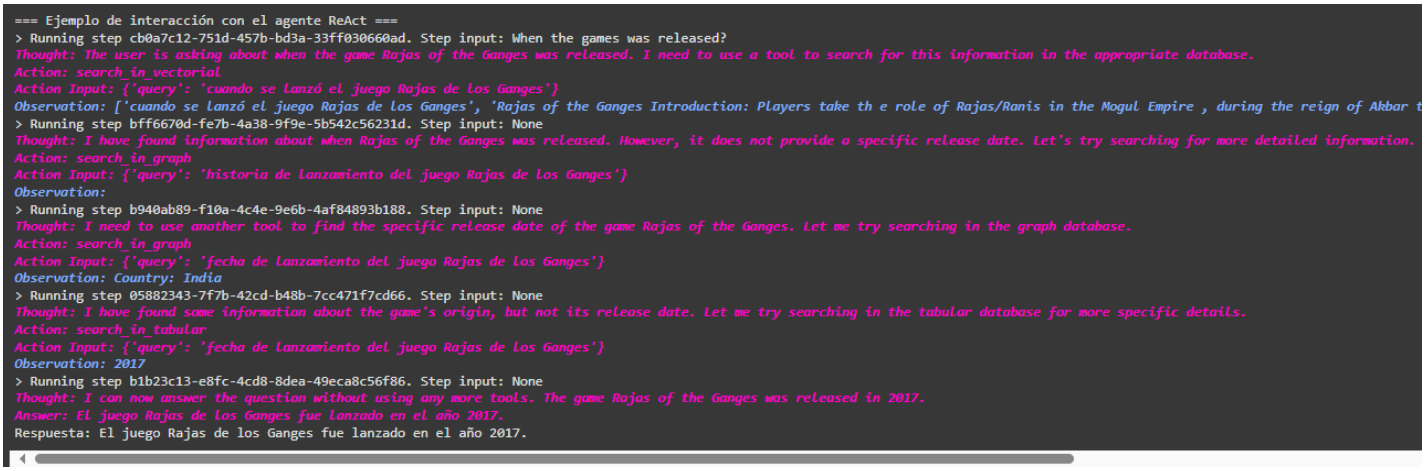
En este ejemplo, lo mismo que el anterior, busca únicamente en la base vectorial y pasa una respuesta basándose en que no existe esa información.

How the portuguese works in the game?



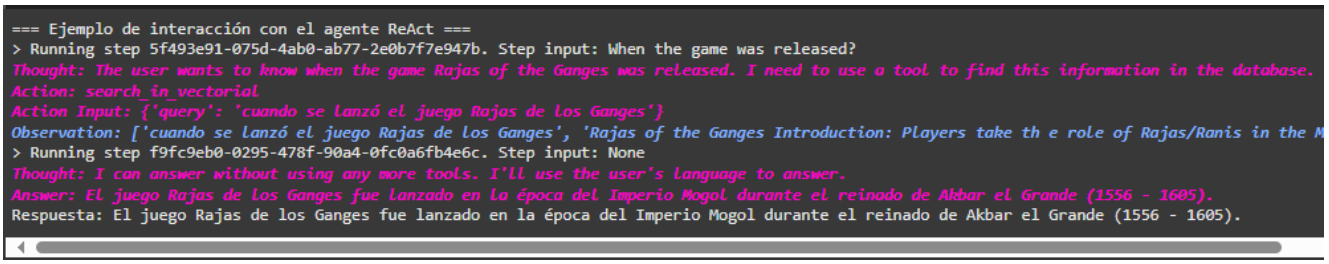
En preguntas que necesitan información de la base de datos vectorial parece tener un buen rendimiento, ya que es lo único que sabe ir a buscar.

When the games was released?



En este caso, tuve un problema de tipeo y obtuve una respuesta correcta, donde el modelo fue a buscar en las diferentes bases de datos. No me parece un funcionamiento correcto.

When the game was released?



Al corregir la pregunta anterior, el agente parece delirar o utilizar únicamente la base vectorial como en las otras preguntas.

Este modelo en general parece tener un problema de no buscar en más herramientas, aun así no sabiendo la respuesta, siendo que en el prompt se le indica lo contrario. Mi hipótesis es que no debe entender bien las reglas o instrucciones del prompt, lo que lo lleva a tener este rendimiento.

gemma:7b

Who are the designers of the game?

```
=== Ejemplo de interacción con el agente ReAct ===
> Running step 1403ee69-28c6-414a-9f5a-12fe22c3aaa1. Step input: who are the designers of the game?
Observation: Error: Could not parse output. Please follow the thought-action-input format. Try again.
> Running step 6b7acead-5e23-4110-a6fe-810e5468b1dc. Step input: None
Thought: The current language of the user is: english. I need to use a tool to help me answer the question.
Action: search_in_vectorial
Action Input: {'properties': AttributedDict([('query', AttributedDict([('title', 'Rajas of the Ganges designers')]))]), 'required': ['query'], 'type': 'object
Observation: Error: search_in_vectorial() got an unexpected keyword argument 'properties'
> Running step b71f96cc-e36e-4c35-8c34-123dc7126367. Step input: None
Thought: I cannot answer the question with the provided tools.
Answer: I do not have access to information about Rajas of the Ganges designers.
Respuesta: I do not have access to information about Rajas of the Ganges designers.
```

Por lo que vemos, este modelo no comprende como utilizar las herramientas y termina por responder que no tiene acceso a la información.

How the portuguese works in the game?

```
=== Ejemplo de interacción con el agente ReAct ===
> Running step b17400ea-0f37-4ca3-b3d4-78d2ce9728f9. Step input: how the portuguese works in the game?
Observation: Error: Could not parse output. Please follow the thought-action-input format. Try again.
> Running step 74dfc21c-37ab-494d-a0a1-bbed088e8cfc. Step input: None
Observation: Error: Could not parse output. Please follow the thought-action-input format. Try again.
> Running step 00797a01-b334-495b-b14c-39ebb2db0d9e. Step input: None
Thought: The current language of the user is Portuguese. I need to use a tool to help me answer the question.
Action: search_in_vectorial
Action Input: {'properties': AttributedDict([('query', AttributedDict([('title', 'Como funcionam o português no jogo')]))]), 'required': ['query'], 'type': 'object'}
Observation: Error: search_in_vectorial() got an unexpected keyword argument 'properties'
> Running step 50394463-4d30-4537-8527-d650fcff6a3a. Step input: None
Thought: I cannot answer the question with the provided tools.
Answer: I am unable to provide information regarding the Portuguese language mechanics in Rajas of the Ganges.
Respuesta: I am unable to provide information regarding the Portuguese language mechanics in Rajas of the Ganges.
```

Probando con preguntas que tengan el contexto en la base de datos vectorial no funciona tampoco, no entiendo cuál puede ser el problema.

When the game was released?

```
=== Ejemplo de interacción con el agente ReAct ===
> Running step 9d830bc4-14fd-4278-b401-12a9832161bf. Step input: when the game was released?
Observation: Error: Could not parse output. Please follow the thought-action-input format. Try again.
> Running step 2ea399ed-db7f-435c-bece-eca797d74e0b. Step input: None
Observation: Error: Could not parse output. Please follow the thought-action-input format. Try again.
> Running step 24e76671-861b-450a-bdc9-672fc5bb9beb. Step input: None
Thought: The current language of the user is: english. I need to use a tool to help me answer the question.
Action: search_in_tabular
Action Input: {'properties': AttributedDict([('query', AttributedDict([('title', 'Rajas of the Ganges release date')]))]), 'required': ['query'], 'type': 'object'}
Observation: Error: search_in_tabular() got an unexpected keyword argument 'properties'
> Running step 5a1cf07a-1e02-4708-b988-7dea7e208b55. Step input: None
Thought: I cannot answer the question with the provided tools.
Answer: I am unable to retrieve release date information using the available tools.
Respuesta: I am unable to retrieve release date information using the available tools.
```

Observamos que no importa la pregunta, el modelo en sí no sabe utilizar las herramientas y termina respondiendo que no sabe.

Haciendo un balance general de los modelos, puedo afirmar que el que mejor funcionó fue **llama3.2:3b**, no sé si en general todos los modelos tienen una forma de entender los prompts de sistema diferente, por el cual yo tenga que hacer modificaciones extensas en el mismo para el funcionamiento, pero habiendo encontrado un prompt en el cual **llama3.2:3b** mayormente se desempeña bien, no me parecía empezar un prompt desde 0 para los otros, sino que ver como se desempeñaban en el mismo, capaz podría haber probado prompts más básicos o quitando algunas reglas, pero me parece que no tiene sentido hacer esto, ya que desde mi punto de vista un modelo con mayor cantidad de parámetros debería entender mejor estas reglas, no funcionar totalmente peor y obligarme a cambiar todo el prompt, entiendo igual que no todos los modelos tienen un entrenamiento igual, pero tampoco hay alguna guía que indique como hacer los prompts de los diferentes modelos de agentes.

Además, al estar limitado por la cantidad de GPU que pueda utilizar de colab, estas pruebas tienen que ser en un tiempo bastante corto y no me permite hacer todos los cambios que me gustarían hacer, como cambiar el prompt o hacer muchas preguntas más.

Preguntas donde se recurre a más de una herramienta

- What is the best strategy to start the game?
- Who are the designers of the game?
- In what year was the game released?
- How the portuguese mechanic works in the game?
- How the karma mechanic influence the game?

Ejemplos de mal funcionamiento

- What is the portuguese in the game?
- Who are the artist of the game?
- In what country is the game located?

Mejoras para el funcionamiento de las preguntas

Para mejorar este mal funcionamiento, uno tiene que mejorar cómo hace la pregunta al agente, ya que estas preguntas podrían tener sentido en una charla entre personas, pero con un modelo de open-source, el cual tiene un corpus no tan grande, uno necesita ser más específico con las preguntas y al reformularlas esto cambiar.

La primera se puede arreglar formulándola como en las preguntas funcionales: How the portuguese mechanic works in the game?, esto sucede, ya que al utilizar what is the portuguese muchas veces entiende como una persona u objeto, mientras que es una mecánica del juego, entonces al hacer el input para las herramientas, termina buscando otros contenidos.

La segunda también es un tema de formulación, ya que al ser ingresada de esa manera, obtiene la respuesta al buscar en la base de grafos, pero sigue buscando información y cambia la relación de artista a developer o publisher, encontrando más información y agregándola a la respuesta final, como era necesario para el tp, pero este agregado no es de utilidad para la respuesta esperada, por lo que, para mejorar esto, se cambia la pregunta a: What is the name of the person that does the art in the game? La cual funciona mayormente, pero puede tener sus fallos, pero responde solamente acerca de quién es la persona que hacer el arte del juego. También para mejorar el funcionamiento de esta query fue necesario ir a la función de querys dinámicas de los grafos y agregar cierta regla para que la query devuelta no tenga “```” al principio o final de la query y se rompa el código.

Y para la última pregunta, la forma de mejorar esto es cambiando la formulación de la pregunta, ya que esta devuelve una respuesta, pero no es para nada lo que se espera, trae una descripción del juego y no en qué país está localizado el juego. Para esto necesitamos tener una pregunta más específica para que teniendo el nombre del país el agente lo entienda como la respuesta, entonces queda una pregunta así: What is the name of the country where the game is located? Donde aunque busque información en diferentes fuentes y haga una respuesta combinando la información, entiende que lo principal es donde está ubicado el juego de mesa y luego información acerca del juego.

Conclusiones

Mi conclusión en este punto fue que en este ejercicio me pareció un poco más complejo que el anterior en sentido de que tiene que realizarse un ajuste más fino para el funcionamiento del agente, y mayormente no es seguro este funcionamiento. Al dejar más variables en mano del propio modelo LLM, uno tiene que experimentar mucho más con los prompts de sistema, con las preguntas que se le hacen, se podría cambiar el modelo, pero todo esto lleva tiempo, y sin alguna API o recurso compartido por COLAB todo este trabajo se vuelve muy tedioso, ya que estamos hablando de modelos no tan grandes, pero que tienen sus 1b o 3b de parámetro. Sin embargo, todo este proceso de encontrar el prompt ideal para el agente es bastante interesante, cada regla o formato o instrucción que se le pasa al modelo, puede cambiar sustancialmente el resultado de las respuestas. A diferencia del ejercicio anterior, se nota que mayormente uno no le indica que hacer explícitamente al agente, sino que él busca con las herramientas proporcionadas, lo que hace que modelos de menor complejidad tiendan a entender menos y funcionar de manera ambigua. Este punto es capaz en lo que más flaquea, obviamente tampoco se le puede pedir demasiado a un modelo open-source que corre en máquinas no tan potentes, pero te pierde demasiado que el modelo cambie tanto de respuesta entre una ejecución y otra, mayormente sucede que funciona con algunas preguntas en específico, pero tiene sus fallos generales, que no los sé responder y termina siendo como probar ciegamente sobre diferentes preguntas con prompts a los que no estoy tan familiarizado en como funcionan. En el desarrollo no destaque tanto este punto, pero generalmente todas las preguntas que funcionan marcadas anteriormente, están pendiendo de hilo, ya que a mí generalmente me funcionaron, pero hay veces que no. Yo tengo la hipótesis que como marcaba anteriormente, al delegarle tantas tareas al agente, y ser de una potencia media/baja, es más probable que sucedan estos errores de delirio o mal funcionamiento. En el RAG todos estos problemas no se ven, primero porque utilizamos un modelo sumamente mayor, con una cantidad de parámetros grande, y porque uno realiza más tareas por su cuenta para darle el contexto ideal al LLM, a diferencia de aquí que el agente se genera su propio contexto.

Librerías y modelos

Librerías

- **llama_index:** LlamaIndex - Build Knowledge Assistants over your Enterprise Data
- **wikipedia:** wikipedia · PyPI
- **logging:** logging — Logging facility for Python — Python 3.13.1 documentation

Modelos

- **Agente ReAct:** llama3.2