UNIVERSIDAD NACIONAL DE ROSARIO

FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento de Lenguaje Natural

Informe sobre Trabajo Práctico Integrador

2024

Francisco Domingo

Índice

1. **Introducción**
2. **Ejercicio 1**: Chatbot experto en Rajas of the Ganges

Resumen

Desarrollo de la solución

Conclusiones

Enlaces

1. **Ejercicio** **2**: Agente Inteligente

Resumen

Desarrollo de la solución

Conclusiones

Enlaces

1. **Conclusiones** **generales**

3

4

4

x

4

5

6

7

8

9

10

11

1. **Introducción**:

En el marco de la materia Procesamiento del Lenguaje Natural, se desarrolló un trabajo práctico enfocado en la implementación de un chatbot experto en el juego de mesa “**Rajas of the Ganges**” estilo Eurogame, utilizando la técnica RAG (Retrieval Augmented Generation). Este proyecto, que constituye el Ejercicio 1, tiene como objetivo principal integrar diversas fuentes de conocimiento (documentos de texto, datos tabulares y bases de datos de grafos) para generar respuestas precisas y contextualmente relevantes en lenguaje español o inglés, alineadas con las consultas de los usuarios.

El proyecto plantea desafíos técnicos significativos, que incluyen el procesamiento de grandes volúmenes de texto (más de 100 páginas), la creación de embeddings para representar datos de manera eficiente, y el diseño de clasificadores para determinar dinámicamente la fuente de información más adecuada en cada caso. Asimismo, se enfatiza la importancia de optimizar las consultas a las bases de datos para garantizar la eficiencia y evitar el uso excesivo de recursos computacionales.

Posteriormente, el Ejercicio 2 propone desarrollar un agente basado en el concepto ReAct (Reasoning + Acting). Este agente utiliza las herramientas antes programadas para realizar búsquedas en documentos, bases de datos de grafos y datos tabulares, además de integrar mecanismos avanzados como prompts diseñados para coordinar estas herramientas y mejorar la precisión de las respuestas.

El presente informe detalla el proceso de desarrollo, desde la preparación de las fuentes de datos y la implementación de los algoritmos de búsqueda y clasificación, hasta el diseño del agente y su evaluación mediante ejemplos prácticos. Además, se identifican las limitaciones del sistema y se proponen posibles mejoras para futuras iteraciones. Este trabajo no solo demuestra el aprendizaje de conceptos avanzados en inteligencia artificial, sino que también ofrece un enfoque práctico y sistemático para resolver problemas complejos en el ámbito de los sistemas conversacionales.

1. **Ejercicio 1**:

2.1- **Resumen**

Para abordar el desarrollo del chatbot experto en el juego mencionado, el trabajo se estructuró en cuatro pasos fundamentales, cada uno diseñado para cumplir con objetivos específicos dentro del proceso:

**Step 1**: Descarga del Contexto (Context Download)

En este primer paso, se recopilaron y descargaron todos los archivos necesarios que constituirán las fuentes de información del chatbot, abarcando documentos de texto, datos tabulares y bases de datos de grafos.

**Step 2**: Clasificador (Classifier)

Este paso consistió en la comparación y evaluación de dos clasificadores, diseñados para categorizar las consultas del usuario en categorías predefinidas. El objetivo fue optimizar la selección del contexto más relevante para cada pregunta.

**Step 3**: Recuperador (Retriever)

En esta etapa, se implementaron los mecanismos de recuperación de información (retrievers) y el reranker, asegurando una búsqueda eficiente y relevante dentro de la base de datos vectorial.

**Step 4**: Preparación del Prompt (Prompt Preparation)

Finalmente, se trabajó en la adaptación del chatbot para manejar consultas en múltiples idiomas, la generación de prompts adecuados y la integración de las funciones de recuperación de información necesarias para responder con precisión a las consultas del usuario.

Esta metodología permitió descomponer un problema complejo en etapas manejables y bien definidas, logrando un desarrollo eficiente y organizado del chatbot.

2.2 **Desarrollo de la Solución**

Para garantizar que nuestro chatbot funcione de manera eficiente tanto en inglés como en español, decidí desarrollar todo el backend en inglés, ya que es un idioma con una estructura semántica y gramatical más sencilla. Además, implementé un moderador de lenguaje que se encargará de realizar las traducciones cuando sea necesario. Esta estrategia no solo facilita la implementación, sino que también optimiza el rendimiento de los modelos de LLM.

**Step 1**

Para comenzar a construir nuestra base de datos vectorial de texto, descargamos y archivamos en Google Drive diversos archivos desde la web de BGG, seleccionando aquellos en inglés que se refieren principalmente a las reglas del juego. Los archivos elegidos fueron:

Rajas\_of\_the\_Ganges\_-\_rules\_in\_brief\_v1.pdf

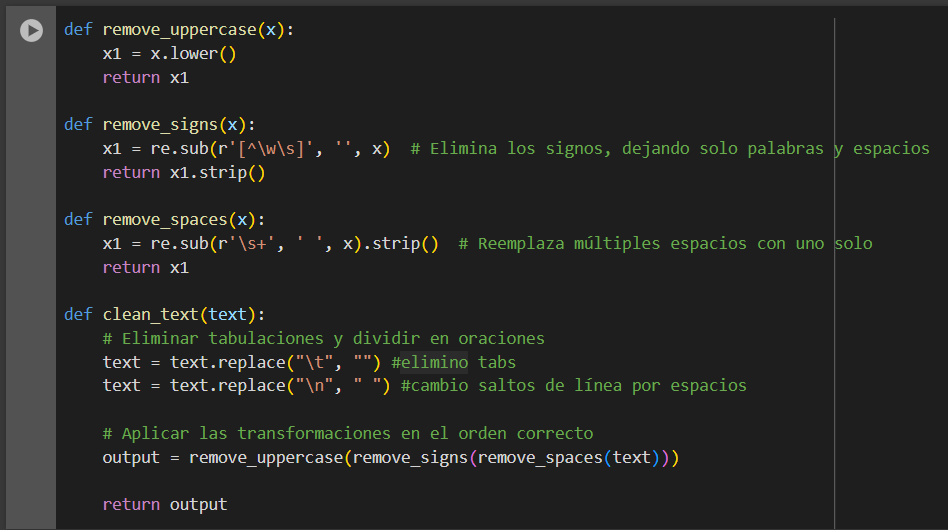
Rajas\_of\_the\_Ganges\_Plain\_and\_Simple.pdf

Rajas\_of\_the\_Ganges\_Quick\_Rules\_Guide.pdf

Rajas\_Player\_Aid\_v1.1.pdf

rules\_EN.pdf

El primer paso fue generar una fuente de datos en formato de texto, extrayendo información de las guías rápidas del juego publicadas por la comunidad de Board Game Geek. Todo este texto se almacena en diferentes variables que luego serán utilizadas para crear los chunks. Además, utilizamos la función clean\_text para limpiar las cadenas de caracteres, asegurando que el texto esté libre de cualquier elemento no deseado y listo para su posterior uso.



La función clean\_text realiza varias transformaciones sobre el texto original para normalizarlo y eliminar elementos no deseados. Estas transformaciones incluyen:

**Eliminar mayúsculas**: Convierte todo el texto a minúsculas utilizando la función remove\_uppercase.

**Eliminar signos de puntuación**: Elimina cualquier signo de puntuación, dejando solo palabras y espacios, mediante la función remove\_signs.

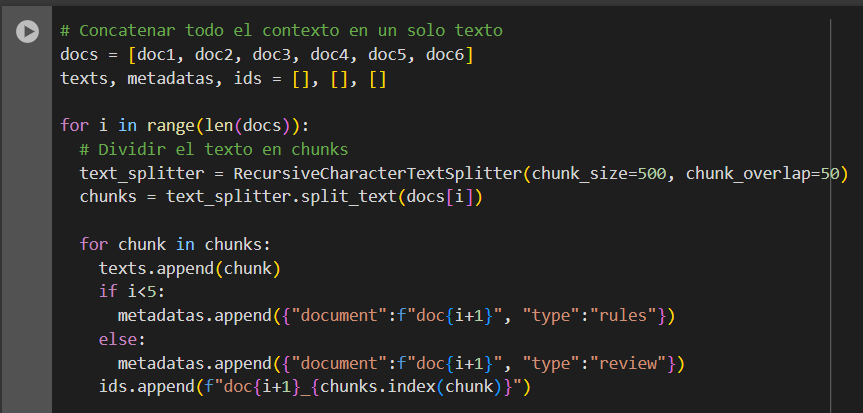
**Eliminar espacios extra**: Reemplaza los espacios múltiples por un solo espacio con la función remove\_spaces.

**Eliminación de tabulaciones** **y saltos de línea**: Sustituye las tabulaciones y saltos de línea por espacios.

El resultado final es un texto limpio y uniforme, listo para ser utilizado en la generación de chunks y su posterior procesamiento.

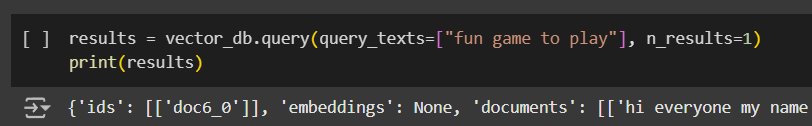
A continuación, procederemos a recuperar las transcripciones de dos videos de YouTube seleccionados, con el objetivo de obtener información sobre opiniones (reviews) del juego. Para ello, utilizamos la librería YouTubeTranscriptApi, aprovechando los métodos que esta proporciona para extraer las transcripciones de los videos.

Con todo el texto obtenido, creamos una lista de chunks de tamaño 500, con un 10% de superposición entre ellos. Estos chunks serán utilizados por el retriever (con el apoyo del reranker) para proporcionar contexto de tipo rules o review en nuestro prompt. Ambas categorías conforman lo que almacenamos como metadata.

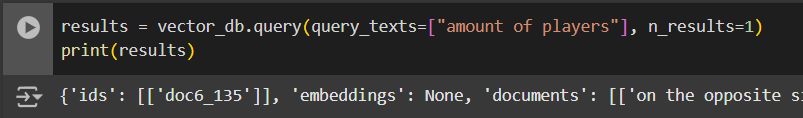


Finalmente, conformamos una base de datos ChromaDB donde almacenamos dichos chunks. Para estos embeddings, utilicé all-MiniLM-L6-v2. Por lo que pude investigar, este modelo es una opción ideal para generar embeddings porque combina buena precisión con alta eficiencia. Es rápido, liviano y genera vectores de baja dimensionalidad (384), lo que lo hace perfecto para proyectos con pocos recursos o donde se necesita procesar textos variados. Además, fue entrenado con datos muy diversos y multilingües, así que funciona bien en un montón de situaciones. Si bien hay modelos más precisos para casos muy específicos o si tenés mucha capacidad computacional, este es una gran elección si buscás algo rápido, versátil y eficiente.

Analizamos qué chunks recupera ChromaDB a partir de cercanía a ciertos tópicos:

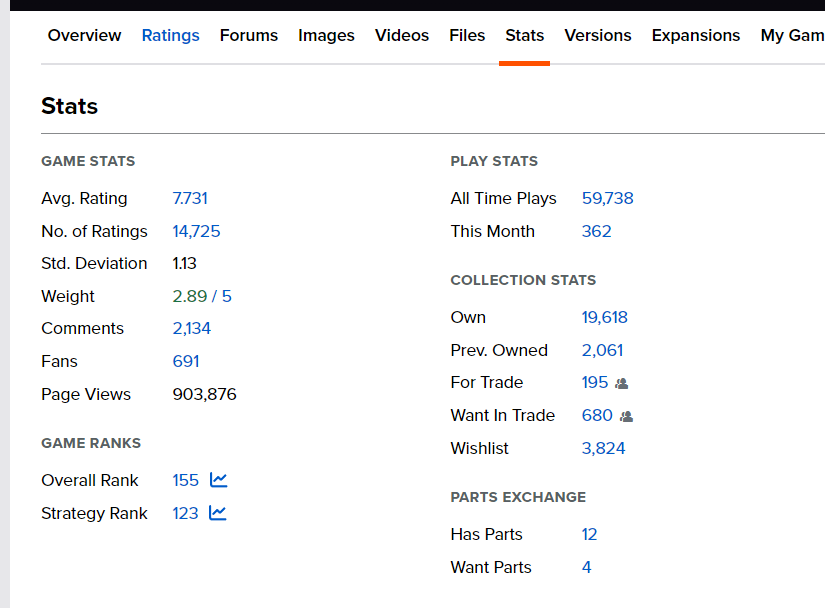


En este ejemplo, proporcionamos la consulta "fun game to play", que está más relacionada con opiniones sobre el juego, y ChromaDB devuelve un chunk del doc6 (transcripción de YouTube). Este resultado es favorable, ya que los videos están clasificados como "review".



En una prueba diferente, el resultado no es tan adecuado, ya que "amount of players" está más relacionado con las reglas del juego. Se espera que, utilizando el clasificador y el retriever con reranking, podamos recuperar en la mayoría de los casos un chunk que responda adecuadamente al prompt proporcionado.

Para la base de datos tabular, hacemos WebScrapping sobre BGG para recuperar la [sección de estadísticas](https://boardgamegeek.com/boardgame/220877/rajas-of-the-ganges/stats). Esta información será clasificada como “stats”.

Para consultar la base de datos de grafos online de [WikiData](https://www.wikidata.org/wiki/Q56759854), definimos dos funciones: **games\_same\_publisher** y **games\_same\_developer**.

La función games\_same\_publisher() realiza una consulta SPARQL a la base de datos de Wikidata para obtener una lista de juegos que han sido publicados por la editorial HUCH!. La consulta utiliza el identificador de la editorial Q1566108 (HUCH!) y busca todos los juegos relacionados con esta editorial.

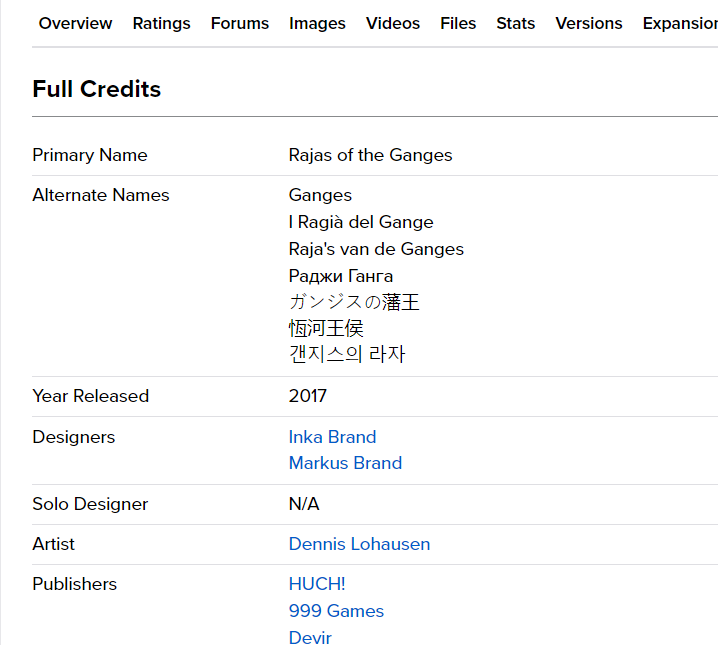
El proceso de la función es el siguiente:

1. Define el punto de acceso SPARQL de Wikidata.
2. Establece una consulta que busca los juegos asociados con la editorial HUCH! (identificada por el URI P123).
3. Realiza una solicitud HTTP GET a Wikidata con los parámetros de la consulta y el formato de respuesta JSON.
4. Si hay resultados, construye una cadena de texto que lista los juegos asociados con HUCH! (su nombre). Si no se encuentran resultados, indica que no se encontraron juegos para esa editorial.
5. Devuelve la lista de juegos publicados por HUCH! o un mensaje que indica que no se encontraron resultados.

Al final, imprime los resultados obtenidos.

La función games\_same\_developer se define de forma similar, pero para poder recuperar aquellos juegos que comparten desarrollador. Esta información será de crucial importancia para poder realizar recomendaciones de juegos en el chatbot.

Para confeccionar la bases de datos de grafos propia, realizamos WebScrapping sobre los [Full Credits](https://boardgamegeek.com/boardgame/220877/rajas-of-the-ganges/credits) provistos por BGG.



Usamos los créditos como categorías de los nodos (Alternate\_Names, Year\_Released, …) y generamos para cada uno de ellos un predicado que nos servirá como arista al nodo central *Rajas of the Ganges*:

"Alternate Names":"can\_be\_translated\_to",

"Year Released":"has\_been\_released\_in",

"Designers":"has\_been\_designed\_by",

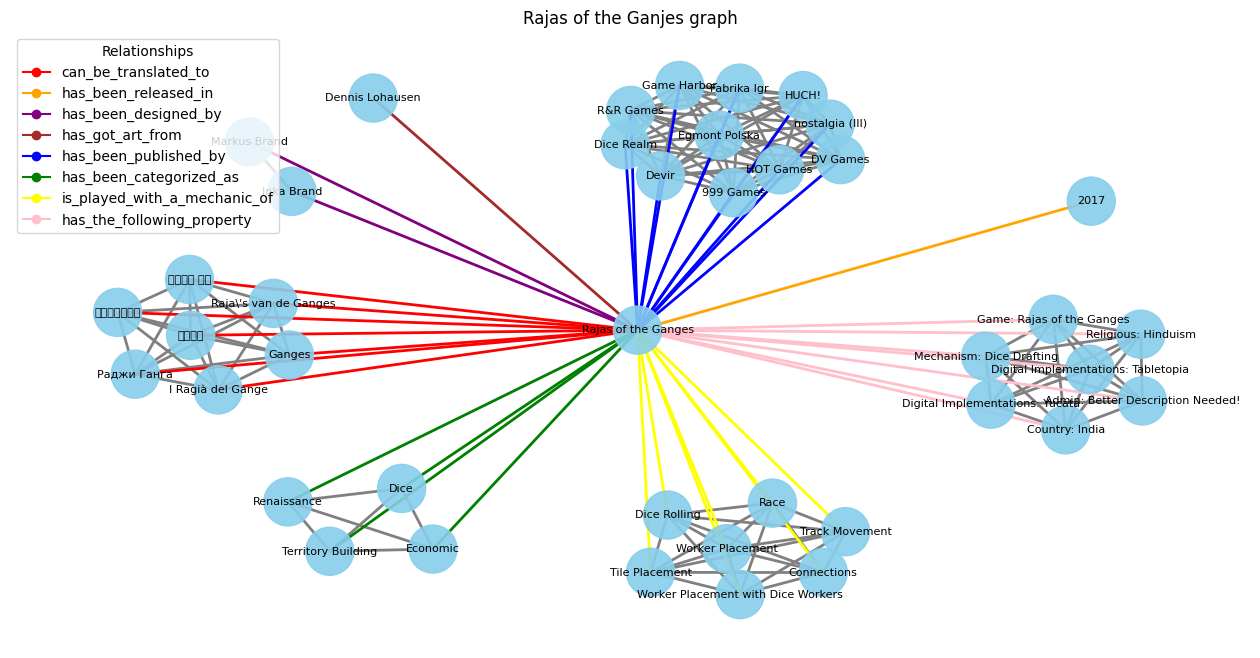
"Artist":"has\_got\_art\_from",

"Publishers":"has\_been\_published\_by",

"Categories":"has\_been\_categorized\_as",

"Mechanisms":"is\_played\_with\_a\_mechanic\_of", "Family":"has\_the\_following\_property"

Con toda esta información, generamos nuestro grafo de redis, sumando además las aristas que conectan nodos de la misma categoría, bajo el concepto de *same\_role*. El grafo resultante es el siguiente:



Como podemos observar, el nodo central representa el juego, y cada uno de los otros nodos un *hecho* que se conecta a este nodo a partir de un predicado (que se puede visualizar en el recuadro de referencia), y con los nodos con los que comparte categoría.

Con esta base de datos conformamos tríadas con el siguiente formato:

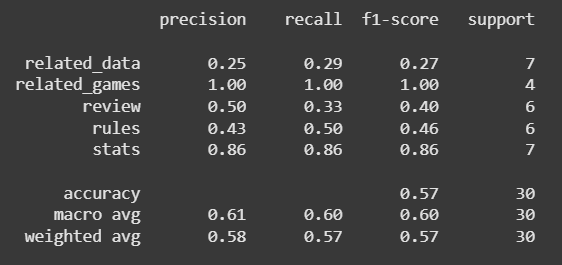
* Rajas of the Ganges can be translated to 恆河王侯
* Rajas of the Ganges has been released in 2017
* Rajas of the Ganges has been published by HUCH!
* Rajas of the Ganges has been designed by Markus Brand
* Rajas of the Ganges has got art from Dennis Lohausen

**Step 2**

Para el clasificador comparamos dos modelos: un modelo de RandomForest y un LLM provisto por HuggingFace.

El modelo de LLM utilizado (Qwen/Qwen2.5-72B-Instruct) es particularmente adecuado para tareas de clasificación complejas, gracias a su enorme capacidad de comprensión. Con 72 mil millones de parámetros, puede captar detalles y matices que otros modelos más pequeños no logran. Además, está diseñado específicamente para seguir instrucciones de manera precisa, lo que lo convierte en la opción ideal para manejar prompts complejos y variados.

Para determinar qué algoritmo sería más adecuado (Random Forest vs LLM), generé datos de prueba utilizando ChatGPT y evalué el rendimiento en términos de precisión (por clase) y accuracy (global).



Mientras que el modelo Random Forest alcanza una precisión del 57%, el modelo LLM logra un 75% en sus ejecuciones. Además, observamos que el modelo de RandomForest mantiene una precisión por clase muy baja, especialmente para related\_data, review y rules, mientras que nuestro LLM mantuvo los siguientes puntajes: [1.0, 0.43, 0.77, 0.97, 0.6].

Debido a esta diferencia de rendimiento, opté por utilizar el LLM como clasificador.

**Step 3**

Para este paso, configuré un retriever cuyo objetivo es obtener la información más relevante de acuerdo con el prompt introducido. Este proceso se optimiza mediante el uso de un clasificador LLM y un reranker aplicado a la base de datos vectorial, mejorando así la precisión de la información recuperada.

El Reranker es un modelo basado en CrossEncoder, que se encarga de reordenar los resultados recuperados. Este modelo evalúa la relevancia de los documentos en función de la consulta, proporcionando una clasificación más precisa de los nodos (documentos) previamente recuperados de la base de datos. En este caso, el modelo CrossEncoder se utiliza para predecir la relevancia de cada documento frente al prompt, asignando una puntuación a cada uno, lo que permite ordenar los resultados de manera que los más relevantes aparezcan primero. El flujo de trabajo completo de este sistema es el siguiente:

* El retriever de base de datos vectorial consulta la base de datos utilizando el prompt y obtiene un conjunto de resultados. Este conjunto es mayor al necesario, para que luego el reranker pueda seleccionar los mejores documentos.
* El reranker, al recibir los documentos y el prompt, predice la relevancia de cada documento, utilizando el modelo CrossEncoder. Los documentos son entonces reordenados de acuerdo con sus puntuaciones, y se seleccionan los mejores, dependiendo de los parámetros configurados.
* En función de la clasificación del prompt, el retriever determina qué tipo de contexto se debe recuperar. Si el prompt está relacionado con reglas, reseñas, estadísticas o juegos relacionados, el sistema accede a las fuentes más adecuadas, como bases de datos vectoriales, DataFrames o grafos.

Este enfoque asegura que el sistema pueda manejar diferentes tipos de consultas y proporcionar la información más relevante en cada caso, mejorando tanto la precisión como la eficiencia del proceso de recuperación. Realizamos pruebas con las siguientes queries, obteniendo estos resultados:

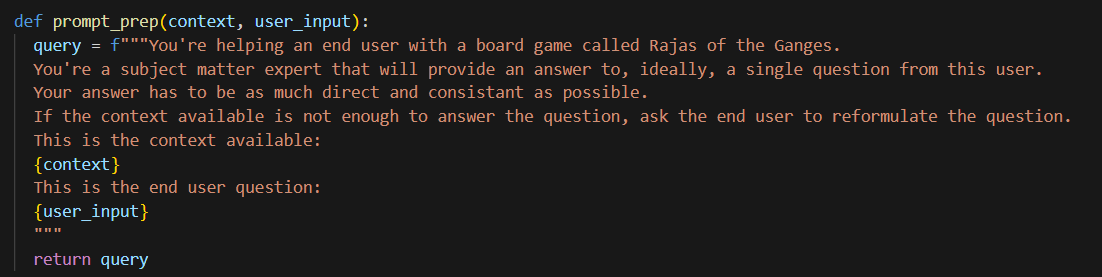
|  |  |  |
| --- | --- | --- |
| **Query** | **Expected** | **Actual** |
| How should I start the game board? | rules | rules |
| How was this game rated by the community? | review | Review |
| What is the game's average rating? | stats | stats |
| How many monthly players does this game have? | stats | stats |
| Can you recommend a similar game? | related\_games | related\_games |
| How is this game called in spanish? | related\_data | related\_data |

Por ello concluimos que nuestro retriever con reranking se comporta de la forma esperada, clasificando correctamente cada una de las preguntas en su clase correspondiente.

**Step 4**

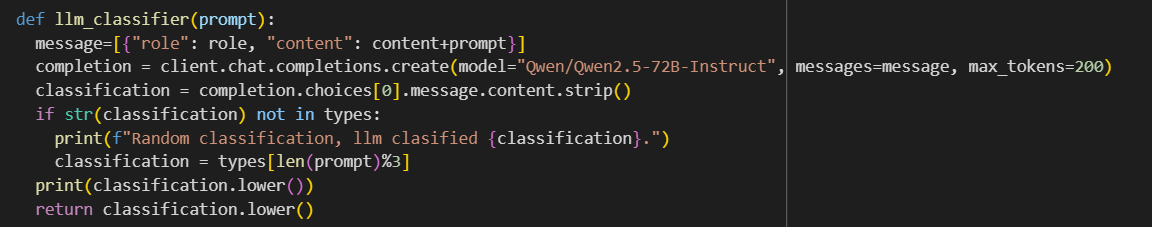
Para esta sección definimos funciones para realizar traducciones inglés-español y español-inglés, haciendo uso de la detección de idioma y de la api de Google Translator, a partir de las librerías que se mencionaron anteriormente.

Después de realizar una extensa experimentación y optimización de prompts, decidí utilizar la siguiente entrada, que considero la más adecuada para el propósito.



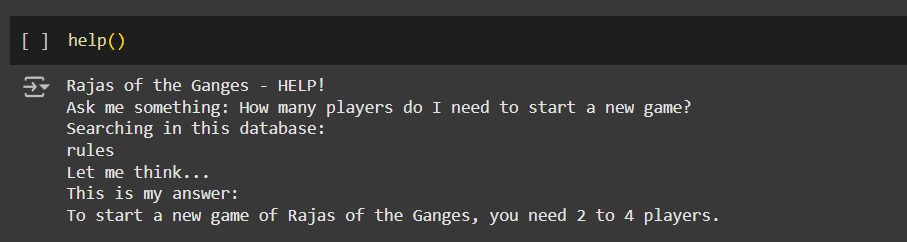
En algunos casos, el motor LLM devolvía caracteres en japonés y símbolos extraños al intentar clasificar el prompt. Esto ocurría porque el clasificador estaba generando respuestas no deseadas, resultado de la instrucción dada de "no agregar caracteres ni ningún otro contexto a las respuestas". Mi objetivo era evitar que el modelo produjera salidas que no correspondieran a los cuatro tipos de clasificación definidos, pero como descubrí, las instrucciones en formato negativo no son efectivas en este caso.

Para solucionar este problema, implementé una solución algo artesanal en la función para garantizar que siempre se devuelva una clase concreta.



Tras ajustar el prompt, el modelo dejó de utilizar la clasificación aleatoria, y la decisión sobre la categoría correcta ahora se toma de manera coherente por parte del LLM.

Se realizaron extensas pruebas del comportamiento del chatbot, tanto manualmente como de forma automatizada, probando de igual manera preguntas en inglés y en español.



Aquí el listado de preguntas-respuestas:

|  |  |
| --- | --- |
| Pregunta | Respuesta |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

2.3 **Conclusiones**

2.4 **Enlaces**

**Fuentes de Datos**:

* La página web “boardgamegeek.com”, que condensa data de todo tipo sobre múltiples juegos similares a éste.
* Guías rápidas de juego desarrolladas por la comunidad.
* Reviews y Gameplays en formato video de la plataforma YouTube.
* Wikidata, como una base de datos de grafos.
* Una base de datos de grafos propia, confeccionada a partir de la página web BGG.

**Librerías**:

* Gdown, para descargar los archivos pdfs localizados en GoogleDrive.
* PyPDF2, para poder recuperar el texto de dichos pdfs.
* Youtube-Transcript-Api, para recuperar las transcripciones de los videos alojados en la plataforma de YouTube.
* Selenium y ChromeDriver, para hacer WebScrapping sobre BGG.
* WikiData, para consultar la base de datos de grafos allí alojada.
* LangChain, para manejar los chunks de forma recursiva.
* Protobuf, Tensorflow y ChromaDB, para trabajar con nuestra base de datos vectorial.
* Redis y RedisGraph, para trabajar con la base de datos de grafo.
* MatPlotLib y Networkx para graficar dichos grafos.
* Langdetect, y Deep\_Translator, para manejar ambos idiomas en nuestro chatbot.

**Ejercicio 2**:

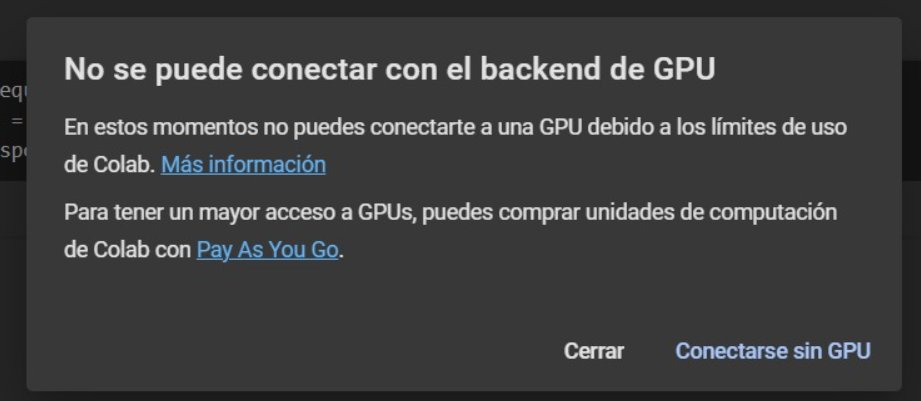
En este otro caso, implementé un sistema basado en un agente de inteligencia artificial diseñado para asistir a usuarios con preguntas relacionadas al juego antes mencionado.

Utilicé el modelo llama3.2, configurado con parámetros como un bajo nivel de aleatoriedad (temperature=0.1) para respuestas más consistentes y una ventana de contexto amplia (context\_window=4000) para manejar consultas complejas. El agente usa las funciones ya definidas (retriever\_vector\_db, retriever\_dataframe\_db y retriever\_graph\_db) para extraer información relevante nuestras bases de datos.

Se utiliza la arquitectura ReAct (Reasoning + Acting), que permite que el modelo analice la consulta, decida qué herramienta usar y actúe en consecuencia. Esto se estructura bajo un formato claro que guía tanto al modelo como a los desarrolladores para mantener consistencia en el flujo de pensamiento, acción y observación.

El agente recibe la consulta del usuario, analiza cómo responder basándose en el contexto y utiliza las herramientas configuradas para encontrar la información adecuada. Si hay un error o una consulta inválida, se maneja la excepción de manera controlada para evitar fallos en el programa.

El código provisto corrió una única vez, luego de reiteradas re-instalaciones de librerías por problemas de incompatibilidades y de ejecución de Ollama sobre GoogleColab. Luego de esa ejecución (cambiando el entorno a tp4), el uso de GPU se acabó para mi cuenta, obteniendo este mensaje de error:



En consecuencia, la ejecución sin GPU finalizó siempre en timeout, incluso aumentando el parámetro a 500 o 1000 segundos. Lamentablemente, no pude probar en profundidad la calidad del código provisto, ni tampoco pude experimentar con las configuraciones del prompt para mejorar la salida.

**Conclusiones**

En este trabajo, implementamos herramientas de procesamiento de lenguaje natural para crear un chatbot y un agente especializado en el juego de mesa "Rajas of the Ganges". A través de diversas fuentes de datos y técnicas avanzadas, como embeddings generados con el modelo all-MiniLM-L6-v2 y consultas SPARQL para bases de datos de grafos, logramos construir un sistema robusto que funciona tanto en inglés como en español. Opté por el modelo de LLM, debido a su alta precisión y capacidad para manejar consultas complejas, superando a alternativas como Random Forest en clasificación. Los resultados obtenidos fueron positivos, destacándose la consistencia en las respuestas. Este trabajo evidencia el potencial del NLP para construir asistentes inteligentes en dominios especializados con contexto.