Trabajo Practico n2

Procesamiento de Lenguaje Natural

Autor: Iair Borgo Elgart

Universidad Nacional de Rosario, TUIA 2024

Fecha de entrega: 18/12/24

Índice

Resumen:

Este trabajo se centra en la creación de un sistema de recuperación de información y respuesta a preguntas para el juego de mesa "Rajas of the Ganges". Este es un *eurogame* ambientado en la antigua India, en donde hasta 4 jugadores se enfrentan en una carrera en acumular poder y riquezas mediante el cuidadoso colocamiento de trabajadores.

El objetivo principal del trabajo es proporcionar a los usuarios una forma eficiente de acceder a información sobre las reglas, mecánicas y otros aspectos del juego. Para lograr esto, se implementó un enfoque híbrido que combina técnicas de procesamiento del lenguaje natural (PLN), búsqueda vectorial, bases de datos de grafos y acceso a información estructurada en tablas. Se utilizaron bibliotecas como spaCy, ChromaDB, Redis Graph y grandes modelos de lenguaje (LLM), específicamente 3 versiones de Qwen, en diversas formas.

1. Introducción

El objetivo principal es proporcionar a los usuarios una herramienta interactiva capaz de ofrecer información precisa y detallada sobre las reglas, mecánicas y estrategias del juego. Para lograr este objetivo, se implementó una arquitectura que combina las técnicas de Recuperación Aumentada por Generación (RAG) y Razonar-Actuar (ReAct), permitiendo al sistema acceder y procesar información de diversas fuentes para generar respuestas contextualmente relevantes.

Los objetivos específicos son:

- Extraer información relevante de diversas fuentes, incluyendo PDFs, foros en línea y la página web del juego.
- Construir una base de conocimiento que permita realizar búsquedas semánticas y relacionales.
- Implementar un sistema de QA que utilice la base de conocimiento para responder a las preguntas de los usuarios de manera precisa y concisa.
- Evaluar el rendimiento del sistema en términos de precisión y relevancia de las respuestas.

2. Implementación

La implementación se realizó en Python, utilizando las siguientes herramientas y bibliotecas:

- Python: Lenguaje de programación principal, junto a paquetes bases para manejo de objetos.

  spaCy: Para procesamiento del lenguaje natural (NLP).

  ChromaDB: Base de datos vectorial para búsqueda semántica.

  Redis (y RedisGraph): Base de datos para almacenamiento y representación de grafos.

  LlamaIndex: Framework para la creación de aplicaciones con LLMs, incluyendo agentes RAG y ReAct.

  Hugging Face Transformers: Para el uso de modelos de lenguaje pre-entrenados.

Selenium y BeautifulSoup: Para web scraping.

PyPDF2: Para el procesamiento de PDFs.

LangChain: Framework para el desarrollo de aplicaciones impulsadas por modelos de lenguaje.

3. Metodología

La base de datos del sistema se construyó a partir de información extraída de las siguientes fuentes:

- BoardGameGeek (BGG): Se recopilaron las reglas oficiales del juego, comentarios de usuarios, reseñas y otros recursos relevantes disponibles en la plataforma. Esto se realizó con la técnica de web scraping, utilizando tanto los paquetes Selenium como BeatifulSoup, para tener la ventaja que cada uno posee.
- Documentos PDF: Ya extraídos del sitio, se procesaron documentos en formato PDF mediante PyPDF2, que contenían información detallada sobre reglas, variantes y estrategias del juego.
- Foros de BGG: Se analizaron hilos de discusión en los foros de BGG para identificar preguntas frecuentes, respuestas y debates sobre el juego. Se utiliza UNICAMENTE la sección de preguntas sobre reglas.

El preprocesamiento de los datos textuales consistió en las siguientes etapas:

- Limpieza: Se eliminaron caracteres especiales, etiquetas HTML y se normalizaron los espacios en blanco para mejorar la consistencia del texto.
- Resumen: Para las preguntas del foro, fueron procesadas a través de un LLM, para recibir en forma resumida la pregunta especifica que tenía el usuario que realizo la consulta y su respuesta.
- Guardado en archivos para su posterior almacenamiento.

En la recolección de los datos se presentaron problemas nuevos más allá del conocimiento que tenía; por ejemplo, para poder acceder a los PDF directamente desde el sitio y no tener que descargarlo aparte se necesitaba tener un previo ingreso de usuario. Para resolver esto se utilizo Selenium, que nos presenta herramientas para poder “clickear” los objetos visibles en la página pudiendo así entrar a una cuenta creada. Otro problema es el mal parseo de los PDF encontrando palabras cortadas. Se intento arreglar utilizando distintas técnicas, pasando de simples RegEx a LLM, pero ninguna resulto muy efectiva. A pesar de esto, el agente destinado a dar respuesta podía comprender la frase sin ningún problema por lo que decidimos que este era un inconveniente menor teniendo en cuenta el tiempo para realizar el trabajo.

Otro de los problemas presentados es que algunos foros tenían un hilo muy grande, por lo que si se realizaba un Split recursivo del texto se iba a perder mucho del significado de la pregunta y la respuesta asociada a esta. Para poder solucionar esto, se decidió pedirle a un modelo de lenguaje (el mismo que luego se utilizara para el agente RAG, Qwen2.5-72b) para que lo resuma, siendo esto una de las cosas en la que tienen mejor desempeño al brindarle toda la información disponible. Se llega al siguiente prompt:

```
sysprompt = f'''You are designed to read a question from a forum and their answers.\
The task you have been assigned is to return, in a maximum of 250 characters,\
a summary with the next format:

This is the first and main question the forum chatter had ? This is the summarized answer

Just use what you have as input, don't invent things you dont know.
If you don't know the answer or there isn't a question, simply return "No information available".
If there is a message from before repeated, it means that a new person is quoting that comment
'''
```

Es necesario destacar un gran problema que surgió en el scrapeo del foro: muchas veces alguien al responder realiza *quoting* sobre lo escrito de otra persona

> **Frank DiLorenzo** @rnrgames wrote:
> Yes, you must select and give up your die with a '2' BEFORE reaping the benefit of the Dancer.

As with any action that requires spending dice.

Esto en el texto ya extraído se presenta como una repetición en un párrafo aparte. Para que el modelo logre “entender” esto se le dice explícitamente que un mensaje repetido significa que se está citando algo anterior. A través de esa simple adición en el prompting se logra mejorar muchas de las síntesis generadas.

Con esto aclarado, el LLM logra realizar síntesis de las preguntas y sus respuestas muy aceptable en la mayoría de los casos, habiendo unos pocos que no se apega al prompt, indicando que falta refinarlo aún más y tal vez mover el hiperparámetro de temperatura

Una vez recolectado todos los datos, se procedió a crear la clase KnowledgeBase que gestiona la base de conocimiento para el almacenamiento, búsqueda y recuperación eficiente de información. Esta base de datos iba a ser luego accedida por un agente RAG o ReAct.

Diseño de la Clase KnowledgeBase

La clase KnowledgeBase fue implementada con una arquitectura modular que combina herramientas de procesamiento semántico, búsqueda vectorial y almacenamiento gráfico. Sus principales componentes son:

1. **Procesamiento de Lenguaje Natural**
    - Se utilizó un procesador de lenguaje, basado en spaCy, para realizar análisis sintáctico y extracción de relaciones semánticas.
    - Este componente permitió identificar entidades clave y relaciones sujetas-verbo-objeto en el texto, lo que resulta esencial para representar las conexiones en el grafo de conocimiento.
2. **Almacenamiento Vectorial**
    - La base de datos semántica se implementó mediante ChromaDB, donde se almacenaron las representaciones vectoriales de los documentos (se utiliza embedding por default MiniLM-L6-v2).

- Los textos fueron divididos en fragmentos mediante el método RecursiveCharacterTextSplitter, permitiendo una indexación más granular, útil para no sobrecargar de tokens la posterior query.

3. **Representación en Grafos**
    - Los datos extraídos se almacenaron en un grafo creado con RedisGraph, utilizando nodos para representar entidades (como términos clave del juego) y relaciones para conectar dichas entidades. Para la extracción de estas relaciones se utilizó el código encontrado en la asignatura, intentando modificarlo para un funcionamiento mas completo. A pesar de esto, no se logro que las entidades extraídas sean perfectas. Una de las alternativas a esto es utilizar aun otra LLM para poder realizar la extracción automática dada los textos, pero no pareció que fuese una decisión económicamente inteligente debido a las restricciones de tokens que nos presentan todas estas en su versión gratuita, y por la restricción de computo que nos presenta Colab. Si pudiera correr un modelo de manera local, puede que sea la forma con mejor flexibilidad y precisión.
4. **Mecanismos de Búsqueda**
    - **Búsqueda Hibrida**: Dentro de ella se utilizan 2 implementaciones más: vector_search que realiza la búsqueda por los embeddings de Chroma; y bm25_search que utiliza el algoritmo BM25 para realizar búsquedas por palabras claves. Una vez obtenido todos los segmentos de los documentos, se reordena los resultados devolviendo hasta el 25% más importante de lo buscado originalmente.
    - **Búsqueda en Grafo**: Se diseñaron consultas Cypher para explorar relaciones entre entidades en el grafo, facilitando la recuperación de conocimientos específicos sobre el juego.
    - **Búsqueda Tabular**: A partir de un DataFrame que almacena metadatos clave, como título, número de jugadores, se implementó un método para obtener información estructurada.
5. **Reordenamiento de Resultados**
    - Para optimizar la relevancia de los resultados, se implementó un reranker basado en el modelo BAAI/bge-reranker-large. Este componente reorganiza los resultados combinando búsquedas vectoriales y por palabras clave, maximizando la precisión de las respuestas, realizando un embedding para la query y el pedazo de información recuperado, y calculando un puntaje de similitud.

Integración y Funcionamiento

La clase KnowledgeBase combina estos componentes para proporcionar un sistema de búsqueda y almacenamiento de información. Al procesar documentos, los datos textuales se pueden limpian, dividir y almacenar tanto en formato vectorial como grafos (dependiendo de los parámetros). Esto permite consultas flexibles, desde preguntas generales hasta búsquedas de relaciones específicas entre entidades.

Además, el sistema garantiza la escalabilidad y modularidad, facilitando futuras extensiones, como la incorporación de nuevas fuentes de datos o mejoras en los modelos de lenguaje utilizados.

En conjunto, este diseño robusto permite gestionar eficientemente el conocimiento sobre el juego y responder a las necesidades de los usuarios de manera precisa y contextualizada.

Pero, para poder utilizar esto construimos arriba **GameExpert,** que es la clase base diseñada para proporcionar respuestas claras y precisas a las consultas relacionadas con las reglas y mecánicas del juego "Rajas of the Ganges". Su objetivo es actuar como un asistente experto en juegos de mesa.

Diseño de la clase GameExpert

La clase **GameExpert** se conecta a un modelo de lenguaje avanzado (LLM) a través del cliente de inferencia de Hugging Face, usando un *token* de autenticación (**HF_TOKEN**) para acceder a la API. El modelo LLM principal utilizado es "Qwen2.5-72B-Instruct".

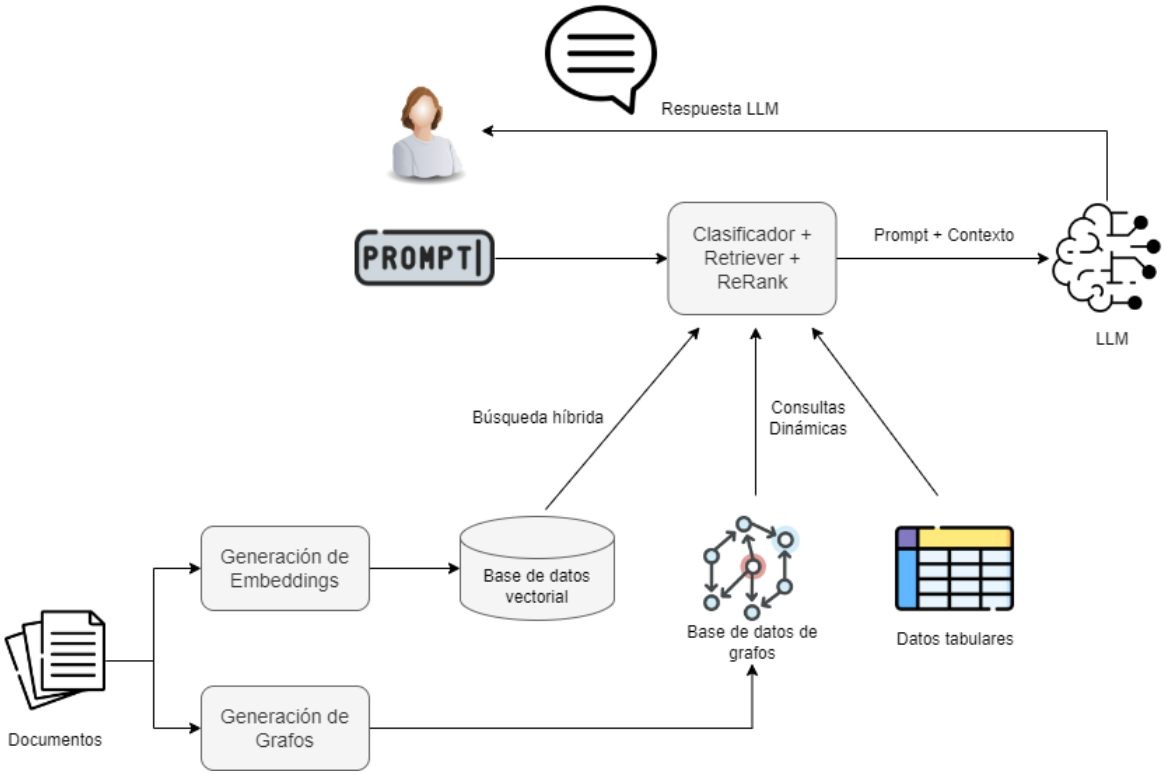
El diseño incluye:

- **Procesamiento de consultas:** Antes de interactuar con el LLM, las consultas se procesan mediante el lenguaje natural, asegurando una representación adecuada en el idioma correcto y adaptando la consulta al formato esperado por el sistema.
- **Prompts específicos:** Un *prompt* del sistema diseñado establece las reglas para la interacción del modelo, enfocándose en:
    - Respuestas claras y directas.
    - Tono natural y conversacional en el idioma del usuario (Ingles o español).
    - Uso de contexto almacenado en la base de conocimiento para respaldar las respuestas.

La función clave process_query() adapta las consultas del usuario para su procesamiento en la base de conocimiento, mientras que update_memory() actualiza el historial conversacional tras cada interacción (aunque, se verá luego que no logra retenerlo).

Esta clase funciona como base para dar la resolución a lo pedido por los ejercicios, heredando sus métodos.

4. Ejercicio 1 - RAGameExpert



Para resolver este ejercicio se tuvo que añadir a la anterior clase una forma de clasificar las queries del usuario para saber a qué base de datos acceder, y qué buscar. Se generaron 50 preguntas y su respectiva base de datos a buscar sintéticamente (a través de un modelo de lenguaje, explicando que contiene cada base de datos y sobre qué juego estamos trabajando) y se realizó separación entrenamiento-testeo.

Para el módulo de clasificación intentó con 2 abordamientos: con regresión logística, utilizando los embeddings generados por Mini-LM-v6 como variables explicativas logrando el siguiente desempeño:

	precision	recall	f1-score	support
document	0.70	1.00	0.82	7
graph	1.00	1.00	1.00	1
tabular	0.00	0.00	0.00	3
accuracy			0.73	11
macro avg	0.57	0.67	0.61	11
weighted avg	0.54	0.73	0.61	11

Y clasificándolo a través de una LLM (Qwen2.5-72b) con un prompt construido específicamente para esta tarea, así su desempeño es satisfactorio. En este caso, el dataset de entrenamiento no se utiliza para realizar few-shot learning ya que logra entender perfectamente lo que se le pide, si no para checkear que en esas 39 preguntas no devuelva algo fuera de lo establecido.

```
classify_prompt = """Classify into 'document', 'graph', or 'tabular'
        You should answer 'document' if the query is about rules or general question of the gameplay.
        You should answer 'graph' if the query is about very basic one-word relations.
        You should answer 'tabular' if the query is about {cols}].
        For most answers, 'document' is okay. Remember to only write ONE WORD, WITHOUT EXPLANATIONS OR QUOTATIONS.
        --------------------------------.
        QUERY:
        {query}

        --------------------------------.
        CLASSIFICATION:

        """
```

Este último logra unas mejores métricas de clasificación a lo largo de múltiples intentos:

	precision	recall	f1-score	support
document	0.88	1.00	0.93	7
graph	1.00	1.00	1.00	1
tabular	1.00	0.67	0.80	3
accuracy			0.91	11
macro avg	0.96	0.89	0.91	11
weighted avg	0.92	0.91	0.90	11

Se decidió utilizar la respuesta generada por el modelo de lenguaje, a pesar del no determinismo de este, ya que en promedio fue mejor y demostró tener una baja variabilidad en las respuestas. En cambio, una regresión es totalmente determinística pero no se logra tener el desempeño de este último. Se podría llegar a un punto medio en el que no es necesario utilizar un modelo tan complejo como lo es el modelo de lenguaje (que, además, se requirió de varias iteraciones en el prompteo para poder lograr ese desempeño) pero al mismo tiempo sea mínimamente más complejo, tal vez uno no lineal, para poder tener una decisión más optima en el espacio 384 dimensional que nos brindó el embedding usado.

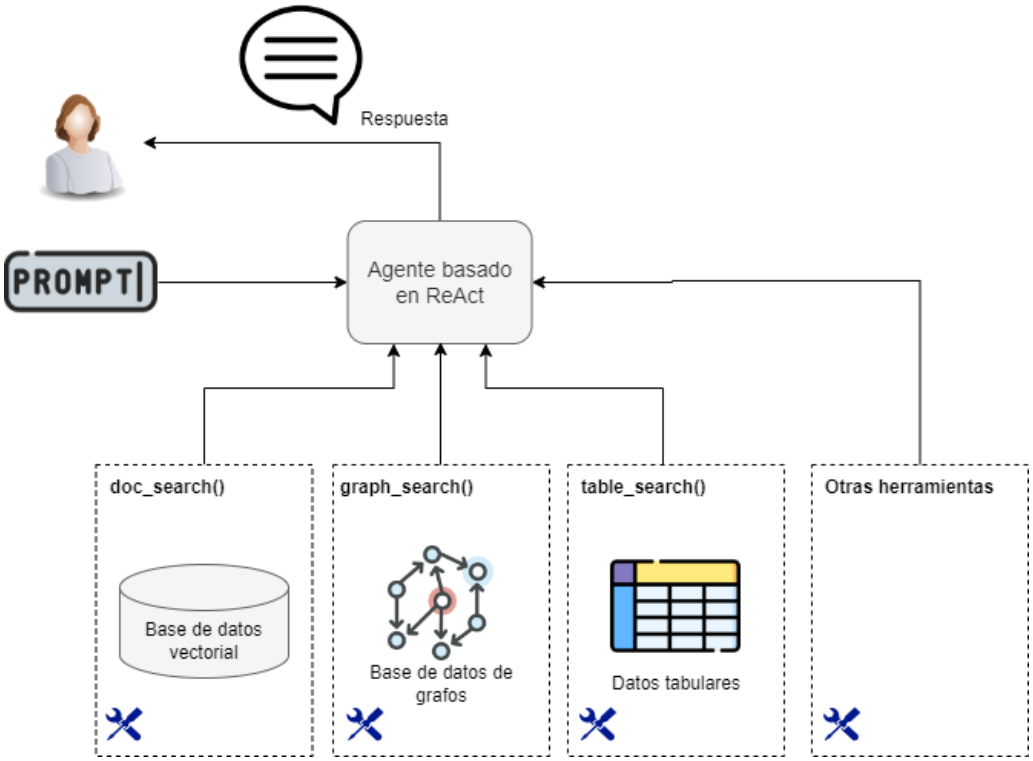
RAGameExpert, una extensión de la clase **GameExpert**, introduce un enfoque híbrido de recuperación de información basado en **Recuperación Aumentada (Retrieval-Augmented Generation, RAG)**. Este paradigma combina búsquedas en bases de datos con modelos generativos, permitiendo respuestas más precisas y contextualizadas. Habiendo ya dicho anteriormente como vamos a realizar la clasificación y lo que nos devuelve; el agente:

1. **Consulta bases de datos:** Dependiendo del tipo de consulta, **RAGameExpert** interactúa con:
    - **Documentos:** Busca información relevante en formato vectorial o de palabras clave.
    - **Grafo:** Utiliza Cypher para extraer relaciones específicas desde el grafo.
    - **Tablas:** Busca información directamente en los datos tabulares mediante *pandas*.

La función _llm_query() se encarga de generar consultas específicas para estas bases de datos utilizando prompts personalizados para cada una.

2. **Reformula la consulta mediante LLM:** Antes de acceder a la base de datos, **RAGameExpert** adapta las consultas mediante el modelo "Qwen2.5-Coder-32B-Instruct", lo que permite obtener los datos en cada lenguaje especifico de las bases de datos.
3. **Llama al modelo LLM con contexto enriquecido:** La función _call_llm() asegura que cada respuesta generada esté respaldada por datos adicionales obtenidos desde las bases de datos consultadas, siguiendo un formato conversacional y basado en lo obtenido.
4. **Rechaza información no disponible: RAGameExpert** tiene prohibido dar respuestas especulativas. Si una consulta no puede ser respondida, el asistente responde claramente con: *"No puedo proporcionar una respuesta"*.

5. Ejercicio 2 - Clase ReActGameExpert



La clase **ReActGameExpert** representa un nivel avanzado en la arquitectura del asistente, diseñado para interactuar con múltiples fuentes de información de manera razonable y eficiente. Esta clase implementa el modelo **ReAct (Reasoning + Acting)**, permitiendo al agente combinar razonamiento lógico con acciones específicas, como consultas en bases de datos y recuperación de información, para responder de manera precisa y contextual. Este ya no utiliza un clasificador, si no que a través de técnicas de prompteo y herramientas que le proporcionamos logra recolectar la información necesaria para responder al usuario

A diferencia de su predecesor, esta clase prescinde del uso directo de un token de Hugging Face (**HF_TOKEN**) y utiliza el modelo "qwen2.5:1.5b" a través de **Ollama**, una solución que corre localmente, bajando los costos de utilizar las limitadas consultas (300) que nos otorga HuggingFace en su versión gratuita. Como contra, su inferencia es más lenta y al tener solo 12gb de RAM disponible en Colab no se logra tener el modelo potente que se tenía en la clase anterior.

Características principales:

1. **Configuración del modelo de lenguaje:**
    - **Modelo:** "qwen2.5:1.5b", optimizado para responder consultas complejas y manejar iteraciones con múltiples herramientas.
    - **Parámetros:**
        - Ventana de contexto extendida (4096 tokens) para manejar consultas detalladas.
        - Temperatura baja (0.3) para asegurar respuestas más determinísticas y consistentes.
        - Iteraciones máximas configuradas en 7 para permitir razonamiento iterativo sin llegar a un bucle infinito.
2. **Sistema de herramientas integrado:**
    - Define un wrapper para cada una de las herramientas especializadas para manejar diferentes tipos de consultas.
    - Estos wrappers de las herramientas son configurados mediante FunctionTool, asegurando la integración fluida con el agente.
3. **Prompt del sistema:**
   El *prompt* guía al modelo para usar el formato ReAct, que descompone el flujo de pensamiento en pasos claros:
    - **Thought:** Explica el razonamiento del modelo sobre cómo abordar la consulta.
    - **Action:** Selecciona la herramienta apropiada para extraer información.
    - **Action Input:** Proporciona los datos necesarios para ejecutar la herramienta.
    - **Observation:** Procesa los resultados de las acciones.
    - **Final Answer:** Combina todas las observaciones en una respuesta final.

Además, el prompt refuerza las reglas clave:

- Responder siempre en el idioma del usuario, incluso si las herramientas generan resultados en otro idioma.
- Limitarse a cinco acciones antes de ofrecer una respuesta o admitir que no se cuenta con suficiente información (esto último implementado a que en ciertas ocasiones, por razones desconocidas, ignora el límite duro de 7).

4. **Integración del agente ReAct:**
    - Utiliza **ReActAgent** para coordinar las herramientas y el modelo de lenguaje, procesando cada consulta en pasos iterativos.
    - Configurado para trabajar con un formato de chat especializado, **ReActChatFormatter**, que asegura la correcta interpretación del flujo de acciones.

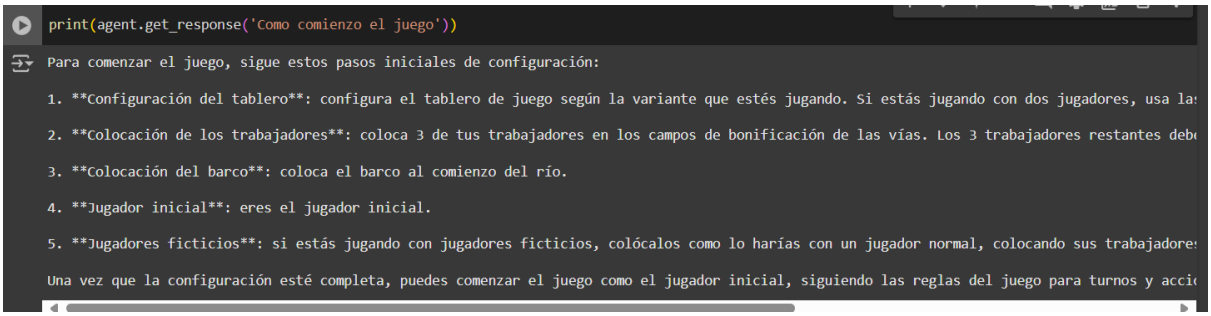
Proceso de consulta en ReActGameExpert

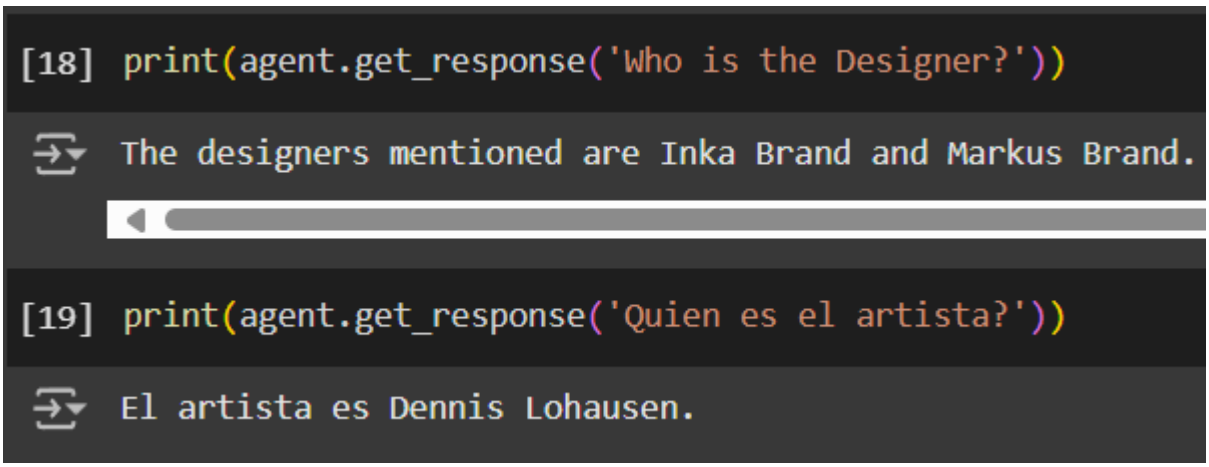
1. **Recepción de la consulta:** La función get_response(query) actúa como punto de entrada, validando si la consulta no está vacía.
2. **Iteración reactiva:**
    - El agente **ReActAgent** descompone la consulta en pasos, seleccionando herramientas y recopilando resultados en cada iteración.
    - Combina observaciones parciales en una respuesta completa.
3. **Gestión de excepciones:** Si ocurre un error durante el procesamiento, el agente devuelve un mensaje de error, asegurando que la experiencia del usuario no se interrumpa.

6. Resultados.

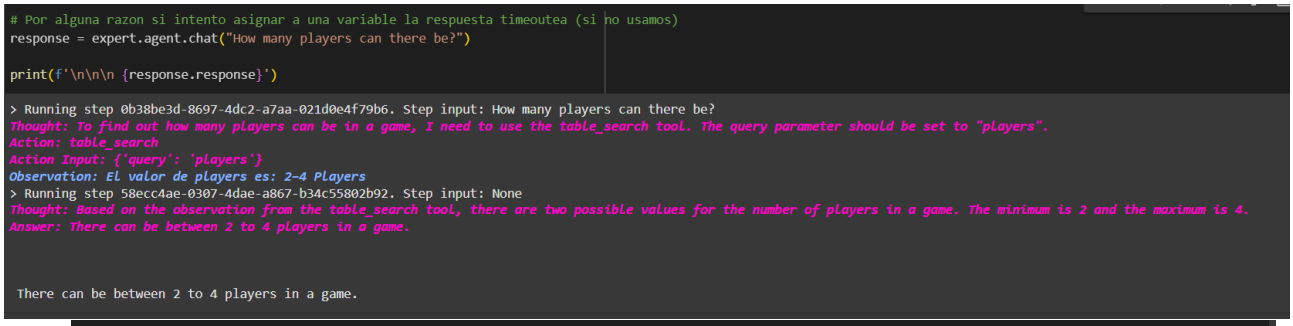
Se evaluó el rendimiento de los agentes RAG y ReAct utilizando un conjunto de preguntas de prueba diseñadas para abarcar diferentes aspectos del juego, desde reglas básicas hasta estrategias avanzadas. Los resultados mostraron lo siguiente:

- RAG: Demostró una alta precisión en la generación de relevantes para preguntas directas y factuales. Sin embargo, tuvo dificultades con preguntas que requerían inferencia o razonamiento complejo sobre múltiples piezas de información. Pudo responder ágilmente tanto a preguntas en español como en inglés, demostrando utilizar aptamente las bases de datos proporcionadas.

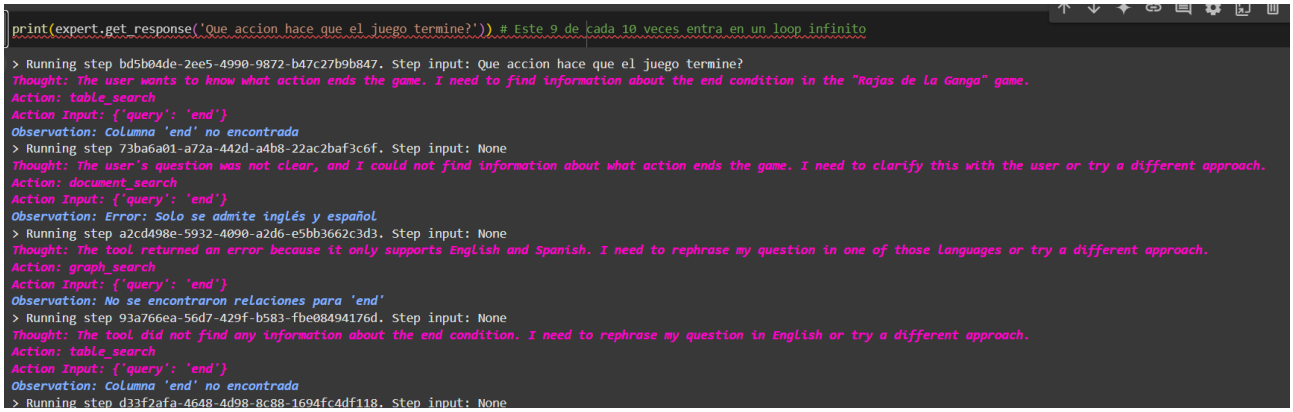




- ReAct: Aunque en papel el funcionamiento es bastante lineal, encontré que este modelo no logra realizar búsquedas buenas, a veces ignorando el limite de iteraciones entrando en bucles infinitos. Intente harcodear con el prompt que la primer base de dato que acceda sea la de documentos pero se rehúsa a hacer caso muchas de las veces.



Ejemplo de loop infinito:



Los dos tienen la peor performance cuando se trata de la base de datos de grafos ya que, a pesar de múltiples intentos de generar una base limpia, las técnicas de NER y reconocimiento de verbo sujeto y objeto crearon un análisis pobre de la estructura del texto. Se hablará de una posible mejora para esto luego.

Aunque en este anterior desarrollo parece que se llegaron a estos 2 agentes finales fácilmente, se requirió releer constantemente tanto documentación como bibliografía de la asignatura para poder llegar a algo que sea mínimamente funcional.

El modelo ReAct al ser implementado de una forma *high-level* no logramos que tenga un razonamiento como es deseable (puede ser que este método inherentemente de muchas vueltas para encontrar una respuesta, según una investigación posterior a terminar este trabajo utilizando el modelo de razonamiento QwQ, también de Qwen, aplica mucha redundancia en su chain-of-thought.) perdiéndose muchas veces en bucles sin poder encontrar la respuesta o con mala utilización de las herramientas a pesar de múltiples iteraciones del prompteo.

7. Conclusiones

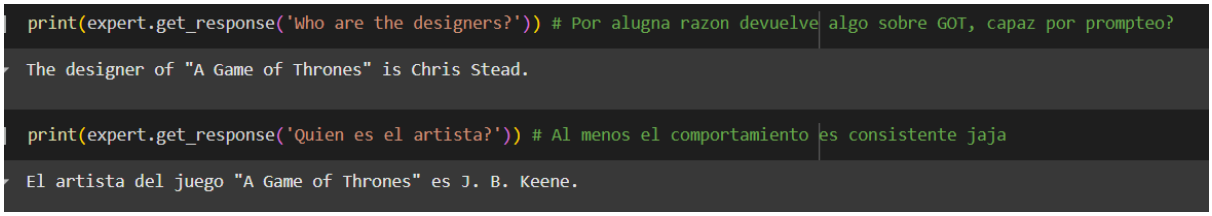
Los resultados obtenidos sugieren que la combinación de RAG y ReAct ofrece un enfoque prometedor para la creación de agentes conversacionales para juegos de mesa. RAG se destaca en la recuperación rápida de información factual, mientras que ReAct proporciona en papel la flexibilidad necesaria para abordar preguntas más complejas. Se podría crear un modelo que primero emplee la simpleza de RAG, limitándose a modelos de regresión logística junto a búsquedas mas simples donde solo se ingresen entidades importadas para una recuperación rápida de la información, y en base a la complejidad de la query del usuario proceder a utilizar un modelo ReAct donde pueda generar las consultas por si solo, a pesar de tener un consumo mayor y una velocidad inferior por la necesidad de pasar la query múltiples veces a través de un LLM.

Las principales limitaciones encontradas:

- La necesidad de un ajuste fino de los modelos de lenguaje y la gestión eficiente de la memoria a largo plazo en las conversaciones, que a pesar de haber intentado una implementación no se logro que mantenga la memoria,
- La mala calidad de la base de datos de grafos. En una idea no implementada, se propuso pasar los documentos a través de (aun otro más) LLM para que pueda

capturar las relaciones mas eficientemente al ser modelos muy flexibles. Esto conlleva un gasto mayor al tener que utilizar nuevamente las costosas pero flexibles arquitecturas que estos modelos presentan, pudiendo ser una idea muy buena si se posee de un equipo con alto poder de computo para uso personal, pudiendo incluso implementar ReAct sobre este para usar enteramente el equipamiento disponible maximizando la calidad de las respuestas.

- Mejorar el prompteo de ReAct: a pesar de reiterarle múltiples veces que tenia un uso especifico, no se logro detener del todos las alucinaciones:



Este trabajo ha demostrado la viabilidad de utilizar técnicas de RAG y ReAct para construir un agente conversacional experto en "Rajas of the Ganges". La combinación de estas técnicas permite aprovechar las fortalezas de cada enfoque, logrando un sistema capaz de responder a una amplia gama de preguntas sobre el juego.

Como trabajos futuros, se propone:

- Explorar técnicas más avanzadas de razonamiento y comprensión del lenguaje natural para mejorar el manejo de preguntas complejas y ambiguas.
- Incorporar modelos de diálogo más sofisticados para mejorar la fluidez y naturalidad de las conversaciones.
- Expandir la base de conocimientos con información adicional sobre estrategias avanzadas, variantes del juego y contenido generado por la comunidad.
- Evaluar el sistema con un conjunto de datos de prueba más extenso y diverso.
- Extender el sistema para que sea aplicable a otros juegos de mesa, ya que la base de datos CSV puede ser fácilmente extendida para varios juegos. Una propuesta interesante es tener múltiples bases vectoriales/grafos para cada juego, que sean accedidas posterior a una búsqueda en la Tabular donde nos indicaría el nombre o dirección donde redirigir la búsqueda. Esto lograría una base unificada para cualquier juego, donde luego se realice una búsqueda mas detallada, optimizando las búsquedas al recortar una gran parte del espacio de embeddings o relaciones.

Como nota final propia, creo que no es necesario la utilización de un modelo ReAct para la simpleza de la mira del trabajo. Este es más adecuado cuando se requiera de razonamientos mas complejos, que una simple lectura de reglas carece

Nota:

A veces HuggingFace tiene las API sobrecargadas y tira este error

```
HTTPError: 500 Server Error: Internal Server Error for url: https://api-inference.huggingface.co/models/Qwen/Qwen2.5-72B-
Instruct/v1/chat/completions

The above exception was the direct cause of the following exception:

HfHubHTTPError                            Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_http.py in hf_raise_for_status(response, endpoint_name)
    475         # Convert `HTTPError` into a `HfHubHTTPError` to display request information
    476         # as well (request id and/or server error message)
--> 477         raise _format(HfHubHTTPError, str(e), response) from e
    478
    479

HfHubHTTPError: 500 Server Error: Internal Server Error for url: https://api-inference.huggingface.co/models/Qwen/Qwen2.5-72B-
Instruct/v1/chat/completions (Request ID: 4uPBzDpVxDtLRnWdHgkMV)

Model too busy, unable to get response in less than 60 second(s)
```

Esperando debería solucionarlo.

8. Bibliografía

- LangChain

LangChain Docs. (s.f.). LangChain Documentation. Disponible en: https://docs.langchain.com/

- SentenceTransformers

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv preprint arXiv:1908.10084. Disponible en: https://www.sbert.net/

- PyPDF2

PyPDF2 Developers. (s.f.). PyPDF2 Documentation. Disponible en: https://pypdf2.readthedocs.io/

- Beautiful Soup

Richardson, L. (s.f.). Beautiful Soup Documentation. Disponible en: https://www.crummy.com/software/BeautifulSoup/

- LlamaIndex

Llama Index Developers. (s.f.). Llama Index Documentation. Disponible en: https://www.llamaindex.ai/

- Selenium

SeleniumHQ. (s.f.). Selenium Documentation. Disponible en: https://www.selenium.dev/

- HuggingFace

Hugging Face. (s.f.). Hugging Face Inference API. Disponible en: https://huggingface.co

- Qwen

Alibaba (cn). Disponible en: https://qwenlm.github.io/blog/qwen2.5/