Trabajo Practico n2

Procesamiento de Lenguaje Natural

Autor: Iair Borgo Elgart

Universidad Nacional de Rosario, TUIA 2024

Fecha de entrega: 18/12/24

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". El objetivo principal 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. 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.

Una vez recolectado todos los datos, se procedió a crear una clase KnowledgeBaseque gestiona la base de conocimiento para el almacenamiento, búsqueda y recuperación eficiente de información.

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.
4. **Mecanismos de Búsqueda**
    - **Búsqueda Hibrida**: Basada en similitudes semánticas y entidades nombradas, mediante ChromaDB, permitiendo encontrar documentos relevantes a partir de consultas en lenguaje natural, con un reranking que será detallado posteriormente
    - **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.

Implementación de 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).

Ampliación con RAGameExpert

RAGameExpert, una extensión de **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. Su implementación incluye características adicionales que completan las capacidades del asistente:

1. **Clasificación del tipo de consulta:** La clase utiliza un modelo de lenguaje (LLM) o, en implementaciones futuras, clasificadores adicionales como regresiones logísticas para determinar el tipo de consulta:
    - **Documentos:** Consultas generales sobre reglas y mecánicas.
    - **Grafo:** Relaciones simples entre entidades como términos específicos del juego.
    - **Tablas:** Consultas sobre columnas específicas en los datos tabulares del juego.

Esta clasificación asegura que cada consulta sea procesada con las herramientas más adecuadas.

2. **Consulta optimizada en 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.

3. **Reformulación de 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.
4. **Llamadas 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.
5. **Rechazo de información no disponible:** Siguiendo principios de diseño ético, **RAGameExpert** tiene prohibido dar respuestas especulativas. Si una consulta no puede ser respondida, el asistente responde claramente con: *"No puedo proporcionar una respuesta"*.

GameExpert 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. Utiliza una base de conocimiento (**KnowledgeBase**) para enriquecer sus capacidades con información textual y contextos procesados previamente.

Clase ReActGameExpert

La clase **ReActGameExpert** representa un nivel avanzado en la arquitectura del asistente, diseñado para interactuar con múltiples fuentes de información de manera reactiva 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.

Diseño de ReActGameExpert

A diferencia de sus 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.

3. Arquitectura de los agentes:

El sistema se compone de los siguientes módulos principales:

- Base de Conocimientos: Se utilizó ChromaDB para almacenar y gestionar los embeddings de los documentos, permitiendo búsquedas semánticas eficientes. RedisGraph se empleó para representar las relaciones entre entidades del juego, como personajes, acciones y recursos.
- Módulo de Búsqueda: Se implementó un sistema de búsqueda híbrida que combina la búsqueda semántica basada en embeddings con la búsqueda por palabras clave, maximizando la recuperación de información relevante.
- Módulo de Clasificación: Se implementó un clasificador basado en un modelo de lenguaje para determinar el tipo de búsqueda óptima (documental, gráfica o tabular) según la pregunta del usuario.
- Modulo de Consulta: Utilizando otra instancia del LLM, se logran las consultas en Cypher o Pandas para recuperar información de las bases de datos.
- Agente RAG: Con lo anterior, se logró un agente RAG que consulta la base de conocimientos y genera respuestas coherentes y contextualmente relevantes.

- Agente ReAct: Se utilizó la biblioteca LlamaIndex para implementar un agente ReAct para realizar las búsquedas complejas en la base de conocimientos o consultas a tablas de datos, mediante la ejecución iterativa de "Pensamiento, Acción, Observación".

4. Implementación

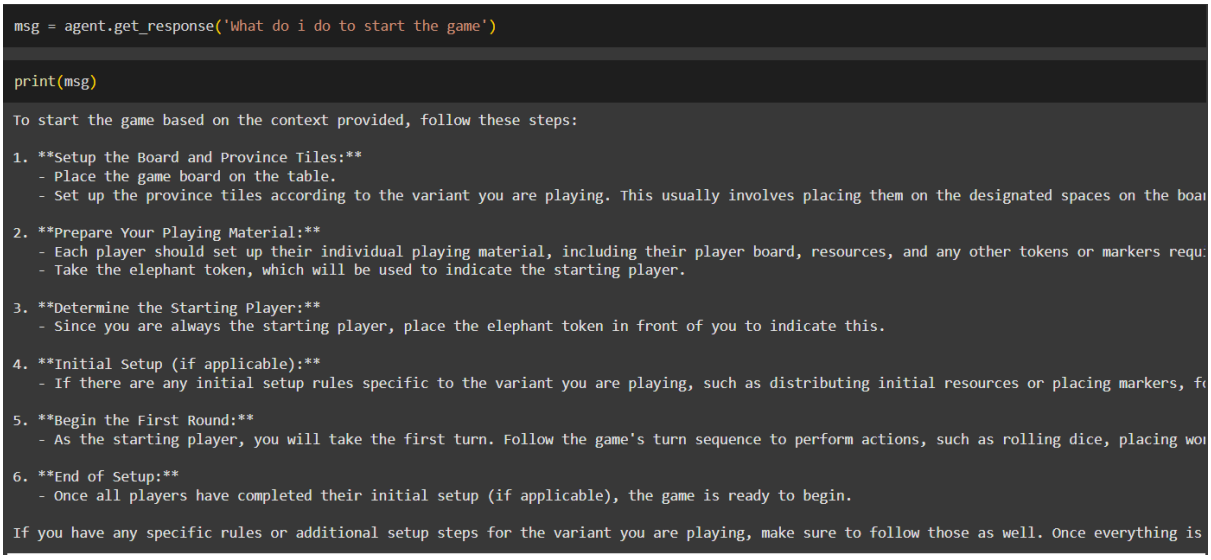
La implementación se realizó en Python, utilizando las siguientes herramientas y bibliotecas:

- Python: Lenguaje de programación principal, junto a paquetes bases para manejo de objetos.
- spaCy: Para el procesamiento del lenguaje natural (tokenización, lematización, etc.).
- ChromaDB: Base de datos vectorial para almacenamiento y búsqueda semántica.
- Redis: Base de datos de grafos para representar relaciones entre entidades.
- LlamaIndex: Framework para la implementación de agentes RAG y ReAct.
- Transformers (Hugging Face): Para el uso de modelos de lenguaje pre-entrenados.
- Selenium y BeautifulSoup: Para la extracción de datos de la web.
- PyPDF2: Para el procesamiento de documentos PDF.

5. Resultados

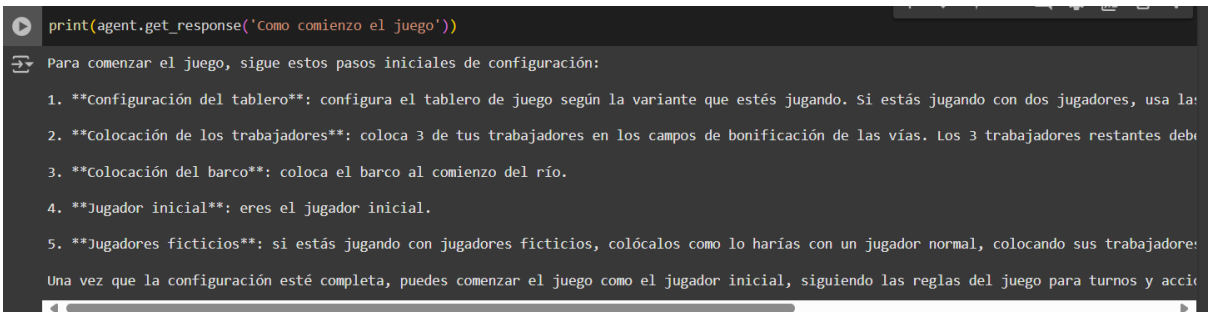
Se evaluó el rendimiento de los agentes RAG y ReAct utilizando un conjunto de preguntas de prueba diseñadas para abarcar diferentes aspectos del juego, desde reglas básicas hasta estrategias avanzadas. Los resultados mostraron lo siguiente:

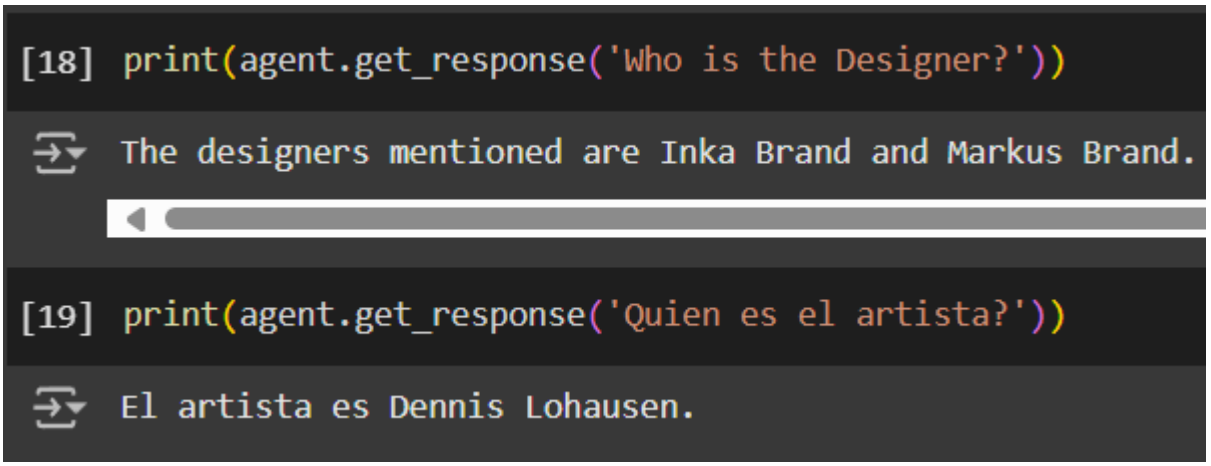
- RAG: Demostró una alta precisión en la generación de relevantes para preguntas directas y factuales. Sin embargo, tuvo dificultades con preguntas que requerían inferencia o razonamiento complejo sobre múltiples piezas de información. Pudo responder ágilmente tanto a preguntas en español como en inglés,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



```
msg = agent.get_response('What do i do to start the game')

print(msg)
```

```
To start the game based on the context provided, follow these steps:

1. **Setup the Board and Province Tiles:**
   - Place the game board on the table.
   - Set up the province tiles according to the variant you are playing. This usually involves placing them on the designated spaces on the boa

2. **Prepare Your Playing Material:**
   - Each player should set up their individual playing material, including their player board, resources, and any other tokens or markers requ
   - Take the elephant token, which will be used to indicate the starting player.

3. **Determine the Starting Player:**
   - Since you are always the starting player, place the elephant token in front of you to indicate this.

4. **Initial Setup (if applicable):**
   - If there are any initial setup rules specific to the variant you are playing, such as distributing initial resources or placing markers, f

5. **Begin the First Round:**
   - As the starting player, you will take the first turn. Follow the game's turn sequence to perform actions, such as rolling dice, placing wo

6. **End of Setup:**
   - Once all players have completed their initial setup (if applicable), the game is ready to begin.

If you have any specific rules or additional setup steps for the variant you are playing, make sure to follow those as well. Once everything is
```





- ReAct: Mostró una mayor capacidad para abordar preguntas complejas que requerían múltiples pasos de razonamiento y acceso a diferentes partes de la base de conocimientos. Sin embargo, el tiempo de respuesta fue generalmente mayor y la precisión fue ligeramente inferior a la de RAG en preguntas más simples. Además, muchas veces intentaba realizar mas pasos de los que eran necesarios

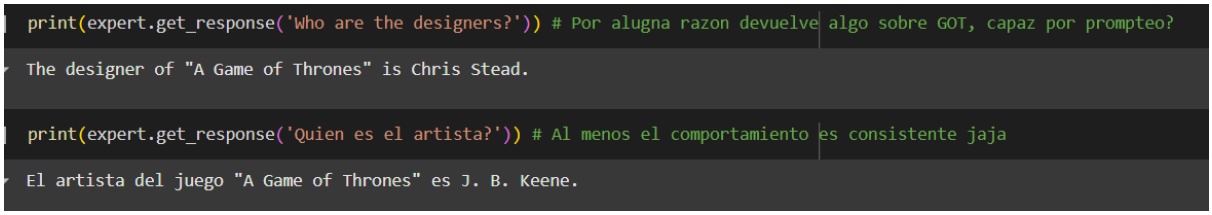
Los dos tienen la peor performance cuando se trata de la base de datos de grafos ya que, a pesar de múltiples intentos de generar una base limpia, las técnicas de NER y reconocimiento de verbo sujeto y objeto crearon un análisis pobre de la estructura del texto. Se hablará de una posible mejora para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

6. 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 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.

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/