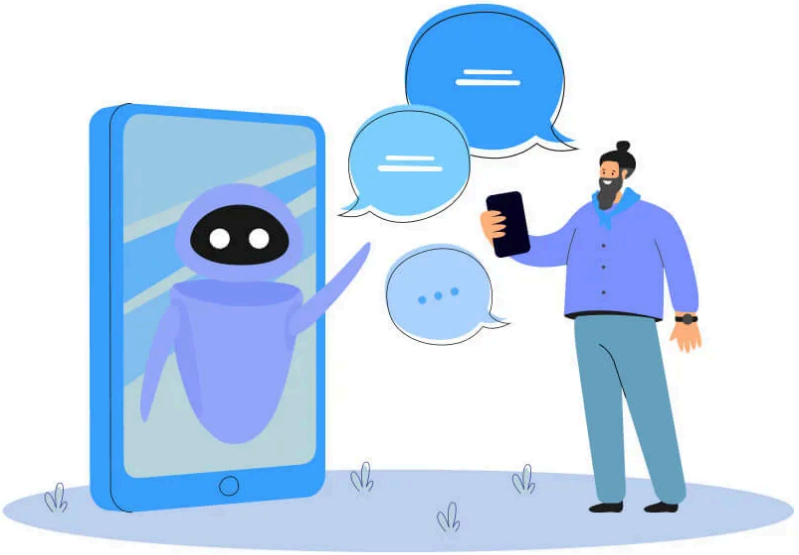
# Trabajo Práctico Final

Procesamiento del Lenguaje Natural



Avecilla Tomás Valentino (A-4239/9)

18 de Diciembre de 2024

Tecnicatura Universitaria en Inteligencia Artificial - UNR

# Índice

# Resumen


## Primera Parte

El primer ejercicio del proyecto aborda el diseño e implementación de un sistema de Recuperación de Información Aumentada (RAG) para asistir en consultas relacionadas con el juego de mesa The White Castle. Los objetivos principales incluyen:

1. Recolectar información relevante de distintas fuentes.
2. Desarrollar distintas bases de datos para almacenar la información.
3. Implementar métodos de recuperación de la información alojada en las distintas bases de datos.
4. Construcción y evaluación del sistema RAG

## Segunda Parte

Por su parte, el ejercicio dos incorpora la idea de Agente el cual se implementó basándose en la primera parte del proyecto y en el concepto ReAct. Los objetivos fueron:

1. Conexión con las bases de datos vía funciones que se basan en los métodos de recuperación usados en el RAG.
2. Desarrollo del Agente con  la librería Llama-Index.
3. Construcción del prompt adecuado y evaluación de resultados

En conjunto, el proyecto demuestra cómo integrar herramientas avanzadas de procesamiento del lenguaje natural y recuperación de información en un entorno especializado. Los resultados destacan la importancia de la flexibilidad en las estrategias de búsqueda y el potencial de los Agentes basados en ReAct para tareas específicas.

# Introducción

## Contexto y Justificación del Trabajo

En el campo del procesamiento del lenguaje natural (PLN), la **Recuperación de Información Aumentada** (RAG) y los agentes inteligentes son tecnologías emergentes con un impacto significativo en la resolución de problemas en dominios específicos. Este proyecto toma como caso práctico el juego de mesa *The White Castle*, un contexto lúdico que plantea retos técnicos al requerir un sistema capaz de recuperar información relevante y específica de diferentes fuentes.

El objetivo principal del proyecto es construir un sistema que proporcione respuestas precisas y contextualizadas, utilizando herramientas avanzadas de PLN.

La relevancia del trabajo radica en que estas herramientas, aunque desarrolladas para un caso particular, son altamente extensibles y adaptables a una variedad de disciplinas y aplicaciones en nuestra sociedad, como la educación o la medicina.

## Objetivos Específicos

1. Diseñar bases de datos optimizadas en formatos tabular, vectorial y de grafos para el almacenamiento y recuperación de información relevante sobre *The White Castle*.
2. Implementar y comparar clasificadores basados en modelos de lenguaje grande (LLMs) y embeddings, evaluando la eficacia de cada enfoque.
3. Desarrollar un sistema de recuperación híbrida que combine búsquedas semánticas y por palabras clave, integrando además un método de ReRank para priorizar los resultados.
4. Crear un sistema de recuperación dinámica que permita interactuar con las bases de datos a través de consultas generadas por LLMs.
5. Implementar un agente basado en el enfoque ReAct, integrando capacidades de recuperación de información y razonamiento para responder consultas complejas.
6. Evaluar el rendimiento del sistema completo en términos de precisión, relevancia y utilidad en el contexto del juego *The White Castle*.

### Estructura del Informe

El informe se organiza en dos grandes secciones:

1. **Desarrollo del Sistema RAG**: Incluye la recolección de datos, la creación de bases de datos híbridas, la implementación de clasificadores y la integración de métodos de recuperación en el sistema.
2. **Implementación del Agente Basado en ReAct**: Explica cómo se amplió el proyecto para incluir un agente inteligente, su interacción con las bases de datos y la evaluación de sus capacidades.

Finalmente, se asume que el lector cuenta con conocimientos técnicos básicos en web scraping para la recolección de información. El foco principal del documento está en la implementación y evaluación del sistema RAG y del agente inteligente.

# Metodología

## Datos Utilizados

El proyecto hace uso de diversas fuentes de datos para el juego *The White Castle*. Estas fuentes incluyen, sobre todo, información extraída mediante web scraping, desde el sitio web propuesto en el enunciado y la respectiva reseña. Puntualmente se buscó extraer información general del juego y opiniones/comentarios de la gente del foro de BGG.

Además desde el sitio web nombrado anteriormente se extrajeron dos archivos desde la sección de 'files' correspondientes al reglamento del juego y a una guía rápida de parte de un jugador.

Para garantizar la calidad de los datos, se implementaron técnicas de limpieza para evitar información dañada. La estructura de los datos recolectados se organizó en formato de documentos (.docx) para facilitar su acceso.

## Métodos y Técnicas Utilizados

En esta sección vamos a recorrer las partes más importantes del proyecto, se buscará comprender la estructura y flujo del mismo, cómo se llevó a cabo y porque se tomaron ciertas decisiones.

## Recolección de Información

La metodología que se siguió para recolectar los datos fue simple:

1. Como primera medida se examinaron los sitios webs ya mencionados donde se podía ver información general del juego *The White Castle*, datos numéricos concretos, un foro y material complementario.
2. Se buscó en el material complementario, concretamente en la sección 'files' del sitio web bgg y se obtuvo un reglamento y un guía "quick start" ambos en formato pdf con imágenes.
3. Se procesaron los archivos, primero para obtener el texto con la libreria **pytesseract,** limpiarlo y normalizarlo.
4. Se inspeccionaron ambas páginas y se recolectó información con web scraping utilizando selenium y beautifulSoup.
    a. Primero desde la página de la reseña donde se buscó todo el texto correspondiente a la misma.
    b. Segundo desde la sección principal del sitio bgg en donde se encontraron todos los datos numéricos que se usaron para construir un dataframe de pandas.
    c. Luego para recolectar información del foro, para esto se buscó crear un script el cual entraba de forma iterativa a cada link del foro y dentro de estos links traía todo el texto de comentarios y opiniones sobre el juego en múltiples idiomas.
5. Por último se almacena todo el texto obtenido (salvo los datos tabulares dentro del dataframe) en tres documentos los cuales se alojan en una carpeta de google drive para no tener que volver a correr el proceso de recolección.
6. Luego se pasa por el último proceso de normalización el cual se encargó de traducir todo el texto al idioma inglés haciendo uso de la librería **translators.**

## Construcción de Bases de Datos

### Base de datos Vectorial

#### *Proceso de Preprocesamiento de los Archivos*

- Lectura de Archivos: Se procesan los documentos recolectados y se extrae su contenido con la función extract_text_from_docx(), que convierte el texto en un formato adecuado para el análisis.
- División en Chunks: Debido a las limitaciones de los modelos de lenguaje (por ejemplo, el modelo de embeddings tiene un límite en la longitud de texto que puede

procesar), el contenido del archivo se divide en fragmentos de texto más pequeños, llamados *chunks*. Cada fragmento se crea a partir de oraciones, utilizando la función dividir_texto_por_oraciones(), que asegura que cada fragmento no exceda los 1000 caracteres para mantener la eficacia y evitar que se trunque el contenido importante.

*Generación de Embeddings*

- Modelo de Embeddings: Para representar el contenido textual de manera numérica, se utiliza un modelo de lenguaje basado en *transformers* (específicamente el modelo multilingual-e5-small, que es capaz de generar embeddings semánticos). Los embeddings son representaciones vectoriales que capturan el significado semántico de cada fragmento de texto.
- Cálculo de Embeddings: La función generar_embeddings() tokeniza cada fragmento de texto y luego utiliza el modelo de *transformers* para calcular los embeddings. Estos embeddings son "promediados" mediante la función average_pool() para obtener una representación unificada del fragmento. Posteriormente, se normalizan para asegurar que las magnitudes de los vectores sean comparables.

*Extracción de Metadatos*

- Reconocimiento de Entidades Nombradas (NER): Usando el modelo de *SpaCy* cargado previamente, se realizan tareas de *Named Entity Recognition* (NER), identificando y extrayendo entidades relevantes dentro del texto. Esto permite clasificar y organizar la información de manera más eficiente.
- Palabras Clave: Además, se extraen palabras clave del texto basadas en su parte del habla (sustantivos y verbos) utilizando el etiquetado de *Part-of-Speech* (POS) de *SpaCy*. Estas palabras clave son seleccionadas en función de su frecuencia y relevancia dentro del contexto del juego.

*Creación de la Base de Datos en ChromaDB*

- ChromaDB: Una vez generados los embeddings y extraídos los metadatos, se utiliza ChromaDB, una base de datos vectorial que permite almacenar y realizar búsquedas eficientes basadas en embeddings. Se crea una nueva colección en ChromaDB (white_castle_embeddings) para almacenar toda la información relacionada con el juego.
- Inserción de Datos: Para cada fragmento de texto, se guarda el fragmento junto con sus embeddings correspondientes y metadatos asociados (como el tipo de documento y las entidades extraídas). Cada entrada en la base de datos se identifica mediante un

identificador único generado aleatoriamente (uuid.uuid4()), asegurando que cada fragmento sea accesible de manera independiente.

*Clasificación de los Fragmentos de Texto*

- Tipo de Documento: Cada archivo es clasificado según su contenido, lo que permite organizar la base de datos en categorías como "reglas", "comentarios", "guía rápida", "reseña", entre otras. Esta clasificación facilita la búsqueda y recuperación de información más relevante en función del tipo de documento.

Todo el contenido de la base de datos está en inglés lo cual es importante porque luego es necesario asegurar que se interactúa con la base de datos en este idioma independientemente del prompt del usuario.

Base de datos de Grafos

*Conexión a la Base de Datos Neo4j*

- Se utiliza la biblioteca neo4j para conectarse a la base de datos Neo4j utilizando las credenciales apropiadas (NEO4J_URI, NEO4J_USERNAME, NEO4J_PASSWORD). Esto permite que el sistema interactúe con la base de datos en la nube de Neo4j y realice operaciones como la creación de nodos y relaciones.
- Creación del Nodo Central: Se crea un nodo central con el nombre *The White Castle* utilizando la consulta Cypher MERGE, que asegura que este nodo se cree solo si no existe previamente.

*Lectura y Fragmentación de Documentos*

- Lectura de Archivos .docx: Los documentos, como los manuales de reglas, reseñas y guías rápidas, se leen desde archivos .docx utilizando la biblioteca python-docx. El contenido de estos archivos se extrae en formato de texto plano.
- Fragmentación del Texto: El texto extraído de cada documento se divide en fragmentos más pequeños o *chunks*. Esto se realiza para que cada fragmento no exceda los 1000 caracteres, lo que es útil para manejar mejor el procesamiento del lenguaje y almacenar la información de manera eficiente.

*Extracción de Tríadas RDF*

- Generación de Tríadas RDF: Para cada fragmento de texto, se utiliza un modelo de lenguaje de Hugging Face (específicamente, el modelo Qwen/Qwen2.5-72B-Instruct) para extraer las tríadas RDF (sujeto, predicado, objeto). Estas tríadas representan las relaciones clave dentro del texto, y el sistema se enfoca en aquellas en las que *The White Castle* es el sujeto o el objeto. Esta tarea se realiza utilizando el modelo de Hugging Face a través del cliente InferenceClient.
- Formato de Salida: El modelo devuelve las tríadas en un formato de texto que se procesa para extraer la relación. Cada tríada representa una relación que involucra *The White Castle* y otros elementos clave (por ejemplo, diseñadores, creadores, relaciones importantes).

*Almacenamiento de Tríadas en Neo4j*

- Parseo de Tríadas RDF: Las tríadas extraídas del modelo se parsean para identificar y separar correctamente el sujeto, el predicado y el objeto. A continuación, se verifica que la tríada involucre al nodo central *The White Castle*, ya sea como sujeto o como objeto.
- Almacenaje en Neo4j: Las tríadas se almacenan en la base de datos utilizando consultas Cypher. Para cada tríada válida, se crean nodos para el sujeto y el objeto, y se establece una relación entre ellos utilizando el predicado. Estas relaciones son de tipo dinámico, es decir, se nombran de acuerdo con el predicado extraído (por ejemplo :DESIGNED_BY, :CREATED_BY, etc.). Los nodos y las relaciones se crean o se actualizan según sea necesario, asegurando que las tripletas se almacenen correctamente en el grafo.

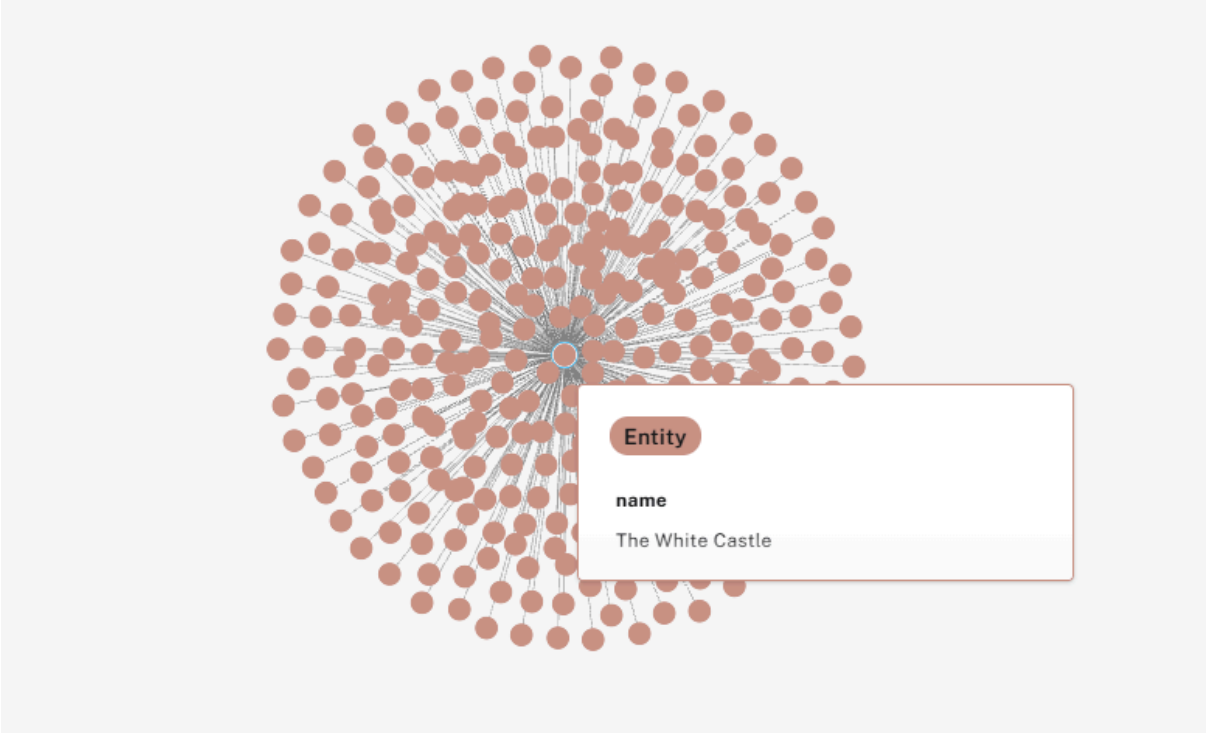
*Procesamiento de Múltiples Archivos*

- Procesamiento Batch de Archivos: Se procesan los documentos. El sistema procesa cada archivo por separado, lee el contenido, lo divide en fragmentos, extrae las tríadas RDF, y las almacena en la base de datos.
- Optimización de Consultas: Gracias al uso de Neo4j, se pueden realizar consultas eficientes sobre las relaciones y nodos almacenados, lo que facilita la exploración de la información y la construcción de un grafo representativo de *The White Castle* y sus elementos clave.

*Resultados*

Habiendo probado varias formas de extraer triadas relevantes sin ‘hardcodear’ se encontró este método muy útil sobre todo por la cantidad de datos con la que se contaba.

El resultado fue una base de datos de grafos con el nodo central correctamente creado y relacionado con el resto, si bien se notó que hubo relaciones irrelevantes, también se encontraron relaciones útiles creadas automáticamente. Podemos concluir que si bien el fuerte de este enfoque es la eficiencia, no deja de ser eficaz.




Captura desde la aplicación Neo4j Aura.

## Métodos de Búsqueda

### DocSearch

*Métodos del constructor y configuración:*

- __init__:
  - Inicializa la clase con el cliente de la base de datos y la colección de documentos.
  - Carga el tokenizador y modelo E5 (de Hugging Face) para generar embeddings semánticos.
  - Configura el dispositivo para usar CUDA si está disponible.
  - Crea un pipeline para el modelo BGE ReRanker, que se utilizará más adelante para mejorar la clasificación de los documentos.

*Métodos auxiliares:*

- average_pool:
    - Calcula el promedio de los embeddings de la última capa del modelo, aplicando una máscara de atención para ignorar los tokens de padding.
- generar_embeddings:
    - Toma un texto, lo tokeniza y genera embeddings semánticos utilizando el modelo E5, los cuales son normalizados antes de ser devueltos.

*Métodos de consulta:*

- rerank:
    - Recibe una consulta y un conjunto de documentos top (por ejemplo, los k más cercanos a la consulta).
    - Usa el modelo BGE ReRanker para predecir la relevancia de la consulta respecto a cada documento.
    - Calcula una nueva puntuación combinando la puntuación de la consulta y la de cada documento.
    - Devuelve los documentos reordenados según su puntuación combinada.
- realizar_consulta:
    - Genera embeddings de la consulta, realiza una búsqueda semántica (usando los embeddings) en la colección de documentos, y devuelve los k resultados más relevantes.

*Métodos adicionales para mejorar los resultados:*

- penalizar_redundancia:
    - Calcula la similitud coseno entre los documentos devueltos y penaliza aquellos que sean demasiado similares entre sí (según un umbral de similitud).
    - Además, penaliza documentos que provengan de un archivo específico (translated_comentarios.docx), reduciendo su puntuación.
    - Garantiza que al menos un documento se mantenga en los resultados, incluso si es redundante.

*Método de búsqueda híbrida:*

- hybrid_search:
    - Realiza una búsqueda semántica (con embeddings) y luego utiliza BM25 para ajustar las puntuaciones de los documentos basados en la coincidencia de palabras clave.

    - Combina las puntuaciones de ambos métodos (semanticidad y palabras clave).
    - Aplica la penalización de redundancia.
    - Reordena los resultados usando el modelo ReRanker.
    - Devuelve una lista de documentos reordenados y penalizados, junto con una cadena con fragmentos de los documentos más relevantes.

*¿Cómo funciona la clase en general?*

- Consulta semántica: La clase busca primero los documentos más relevantes en base a embeddings, utilizando una consulta semántica (realizada por realizar_consulta y generar_embeddings).
- Ajuste con BM25: Se utiliza BM25 para añadir una capa de relevancia basada en palabras clave.
- Redundancia y penalización: Si algunos documentos son demasiado similares, se penalizan para evitar resultados repetidos o redundantes.
- Re-ranking: Finalmente, se usa el modelo BGE ReRanker para volver a ordenar los resultados de acuerdo a su relevancia combinada.

TabularSearch

*Método de inicialización:*

- __init__:
    - Recibe un DataFrame de Pandas como entrada, que contiene información sobre diferentes características de un juego, como calificación, año, número de jugadores, etc.
    - Configura parámetros como la temperature y las stop_sequences para controlar el comportamiento del modelo de lenguaje Qwen.
    - Inicializa un cliente para interactuar con la API de Qwen, que se utiliza para generar consultas tabulares.

*Métodos principales:*

- query_model_for_tabular:
    - Este método envía un prompt al modelo Qwen para generar una consulta tabular.
    - El prompt debe describir lo que se busca en el DataFrame, y el modelo generará una consulta en formato Python que accede a las columnas del DataFrame.

    - El modelo utiliza un contexto que indica las columnas del DataFrame.
    - La salida esperada del modelo es un código de acceso a esas columnas en forma de consulta, que es luego procesado para extraer los resultados.
    - Ejemplo de consulta generada por el modelo: df[df['Rating'] > 4.5]
- tabular_search:
    - Este método recibe un prompt, que es enviado al modelo a través de query_model_for_tabular.
    - La consulta generada es limpiada y adaptada para ser ejecutada en el contexto de la clase.
    - La consulta se evalúa utilizando eval(), que ejecuta el código generado por el modelo para acceder al DataFrame y obtener los resultados.
    - Si la consulta devuelve resultados (es decir, filas del DataFrame), estos se formatean en una cadena y se retornan como una respuesta.
    - Si la consulta no genera resultados, se devuelve un mensaje indicando que no se encontraron resultados.
    - Si ocurre un error en cualquier parte del proceso, se maneja la excepción y se devuelve un mensaje de error.

*Flujo de trabajo:*

1. El usuario proporciona un prompt que describe la consulta que desea realizar sobre el DataFrame (por ejemplo, "Mostrar todos los juegos con una calificación mayor a 4").
2. El modelo Qwen genera una consulta en código Python que accede a las columnas del DataFrame y devuelve las filas correspondientes.
3. La consulta generada se limpia y se ejecuta mediante eval().
4. Los resultados se devuelven al usuario como una cadena de texto formateada.

GraphSearch

*Método de inicialización:*

- __init__:
    - Recibe un graph_client que es un cliente para interactuar con la base de datos de grafos (en este caso, un cliente de Neo4j).
    - Recibe una clave de API (api_key) que se utiliza para autenticar el acceso a la API de Hugging Face.

    - Establece una temperature y stop_sequences para controlar el comportamiento del modelo Qwen.
    - Inicializa un cliente de Qwen para generar las consultas Cypher.

*Métodos principales:*

- query_model_for_cypher:
    - Este método envía un prompt y un pos_context al modelo Qwen para generar una consulta Cypher.
    - El prompt debe ser una descripción de lo que se busca, y el pos_context es un contexto adicional obtenido a partir de análisis POS (Part-of-Speech) de entidades y relaciones en la base de datos.
    - El modelo Qwen genera una consulta Cypher que se utiliza para consultar la base de datos de grafos (por ejemplo, MATCH (twc:Entity)-[:HAS_DESIGNER]->(designer) RETURN designer).
    - La consulta Cypher generada es limpiada y devuelta como una cadena de texto.
- graph_search:
    - Este método toma un prompt y un contexto adicional (pos_context) y utiliza query_model_for_cypher para generar una consulta Cypher.
    - Luego, ejecuta la consulta en la base de datos de grafos utilizando el cliente Neo4j (graph_client).
    - Si se encuentran resultados, se formatean y se devuelven como una cadena. Si no hay resultados, se devuelve un mensaje indicando que no se encontraron datos.
- search_relations_by_pos:
    - Este método realiza una búsqueda de relaciones cuyo nombre contenga una palabra clave extraída de un análisis POS.
    - Se genera una consulta Cypher para buscar relaciones que coincidan con la palabra clave (pos_word) extraída del texto.
    - La consulta busca todas las relaciones cuyo tipo contenga la palabra clave proporcionada y devuelve los resultados encontrados.

*Flujo de trabajo:*

1. El usuario proporciona un prompt que describe lo que desea buscar en el grafo (por ejemplo, "Encuentra el diseñador del juego The White Castle").
2. Se utiliza el modelo Qwen para generar una consulta Cypher basada en el prompt y en el contexto de las relaciones en el grafo.

3. La consulta Cypher generada se ejecuta en la base de datos de grafos (Neo4j) para obtener los resultados.
4. Los resultados de la búsqueda se formatean y se devuelven al usuario como una cadena de texto.
5. En el proceso también se realiza un análisis POS para extraer entidades clave y buscar relaciones que contengan esas entidades, proporcionando contexto adicional.

*Método auxiliar: add_pos_context*

- add_pos_context:
    - Este método realiza un análisis POS sobre el prompt del usuario utilizando spaCy.
    - Extrae las palabras clave de tipo sustantivo, pronombre, adjetivo y verbo.
    - Luego, busca en la base de datos de grafos todas las relaciones que contengan esas palabras clave en su nombre.
    - Agrega el contexto adicional (relaciones encontradas) al texto original del prompt y lo devuelve.
    - Esto proporciona un contexto enriquecido para generar consultas más precisas y relevantes.

## Clasificadores

### Clasificador basado en ejemplos

#### Cargar el modelo de embeddings:

Se carga el modelo de embeddings preentrenado all-mpnet-base-v2 desde la librería SentenceTransformers. Este modelo convierte las frases en vectores de alta dimensión (embeddings) que capturan semánticamente el significado del texto.

#### Dataset de ejemplos:

Se tiene un conjunto de datos (dataset) con ejemplos de preguntas categorizadas en tres grupos:

- Reglas (rules): Preguntas sobre las reglas del juego.
- Reseñas (reviews): Preguntas sobre cómo los jugadores perciben el juego.
- Comentarios (comments): Preguntas sobre estrategias y tácticas para jugar.

Cada entrada tiene un par (label, prompt), donde label es un número (0, 1, 2) que corresponde a las categorías y prompt es la pregunta o declaración.

#### Preparar los datos:

Se separan los datos en dos listas:

- X: Las preguntas (prompts).
- y: Las etiquetas (categorías) correspondientes.

#### Generar embeddings para los prompts:

Cada prompt se convierte en un embedding vectorial utilizando el modelo SentenceTransformer. Esto permite representar los textos en un espacio de alta dimensión para que puedan ser utilizados por el clasificador.

#### Dividir el conjunto de datos:

El conjunto de datos se divide en dos partes:

- Entrenamiento (X_train, y_train).
- Prueba (X_test, y_test). El 80% de los datos se usan para entrenamiento y el 20% restante para evaluación.

#### Entrenar el clasificador:

Se entrena un clasificador de regresión logística utilizando los embeddings generados para los datos de entrenamiento.

#### Evaluar el modelo:

El modelo se evalúa utilizando los datos de prueba. Se imprime la precisión y un reporte de clasificación detallado que incluye métricas como precisión, recall y F1-score.

### Clasificador basado en LLM

#### Método de Inicialización

- model_name: Es el nombre del modelo preentrenado que se utilizará para la clasificación. El valor predeterminado es Qwen/Qwen2.5-72B-Instruct, que es un modelo de Hugging Face.
- context: Es el contexto base que se utiliza para ayudar al modelo a hacer la clasificación. Si no se proporciona, se utiliza un contexto por defecto definido en el método _default_context.
- labels: Son las tres categorías posibles para la clasificación: Documents, Graph, y Table.

- qwen_client: Se inicializa el cliente de inferencia de Hugging Face para interactuar con el modelo Qwen.

Método _default_context

- Este método devuelve un contexto base predefinido que describe brevemente cada una de las tres categorías.
    - Documents: Para preguntas sobre las reglas, estrategias y textos relacionados con el juego.
    - Graph: Para preguntas sobre los creadores, diseñadores e interacciones entre personas.
    - Table: Para preguntas sobre estadísticas del juego como precio, duración, número de jugadores, etc.

Método clasificar

- Este es el método principal que toma un prompt del usuario y lo clasifica en una de las tres categorías mencionadas: Documents, Graph, o Table.
- Flujo:
    1. El método primero verifica si el prompt está vacío. Si está vacío, lanza un error.
    2. Combina el contexto base con el prompt para formar el input_text.
    3. Luego, hace una llamada al modelo Qwen para obtener una clasificación.
    4. La respuesta del modelo se procesa y se devuelve la clasificación.

Uso del InferenceClient

- InferenceClient: Se usa para interactuar con el modelo Qwen. El método chat.completions.create es el que genera la clasificación, pasando el contexto y el prompt del usuario como entrada.
    - Parámetros relevantes:
        - messages: Se pasan dos mensajes: uno del sistema que describe el rol del modelo y otro del usuario con la consulta.
        - temperature: Ajusta la creatividad de la respuesta. En este caso, está configurado en 0.4, lo que significa respuestas menos creativas.
        - stop_sequences: Se utiliza para indicar que la respuesta del modelo debe terminar cuando encuentra "END_RESPONSE".
        - max_tokens: Limita la longitud de la respuesta del modelo.

### Flujo de Ejecución:

- Cuando se llama al método clasificar con un prompt, el sistema prepara el mensaje de entrada, lo envía al modelo para obtener una clasificación, y luego devuelve la categoría que se considera más probable según el modelo.

### Elección de clasificador

La elección final es del clasificador basado en LLM (Qwen) es la adecuada en este caso por varias razones:

1. Capacidad de comprender contextos complejos: El modelo LLM puede manejar preguntas ambiguas y de contexto especializado.
2. Alta precisión en la clasificación de categorías con texto ambiguo o difícil de categorizar de forma simple.
3. Acceso a un modelo avanzado que garantiza un rendimiento actualizado y de última generación.

A pesar de los posibles costos asociados, la precisión y la capacidad de adaptación del modelo LLM en este contexto específico hacen que sea una elección eficiente y sostenible a largo plazo.

## Recuperación de Información Aumentada (RAG)

La metodología de Recuperación Aumentada con Generación (RAG) combina dos fases clave: recuperación de información y generación de respuestas. Este enfoque permite que los sistemas de diálogo, como el que estás desarrollando para el juego *The White Castle*, proporcionan respuestas más precisas y contextualizadas al integrar datos externos durante el proceso de generación. A continuación, se describe el flujo de trabajo en el contexto de tu proyecto.

### Flujo de Trabajo de RAG

1. Entrada del Usuario:
    - El ciclo comienza cuando el usuario ingresa una consulta relacionada con el juego *The White Castle*. Esto puede incluir preguntas sobre reglas, estrategias o detalles específicos del juego.
2. Detección del Idioma:
    - Se detecta el idioma en el que se realiza la consulta utilizando la librería langdetect. Si la consulta está en español, se traduce automáticamente al

inglés para facilitar la interacción con el modelo generativo y otros sistemas de recuperación que operan en inglés.

3. Clasificación del Prompt:
    - La consulta del usuario es pasada a un clasificador que determina a qué categoría pertenece. Las categorías incluyen:
        - Documentos (por ejemplo, reglas, manuales)
        - Grafo (información estructurada sobre relaciones, como reglas interactivas o estrategias)
        - Tablas (datos tabulados, como estadísticas de juego)
4. Dependiendo de la categoría, el sistema elige el método adecuado de recuperación de información.
5. Recuperación de Información:
    - Documentos: Si la consulta pertenece a esta categoría, se utiliza un sistema híbrido de búsqueda documental que permite recuperar fragmentos relevantes del contenido del juego (por ejemplo, reglas o detalles del manual).
    - Grafo: Si la consulta está relacionada con relaciones entre entidades o eventos del juego, se realiza una búsqueda en un grafo de conocimiento que almacena las conexiones y reglas específicas del juego.
    - Tablas: Si se refiere a datos estructurados o estadísticas, el sistema consulta una base de datos tabular.
6. Ajuste del Contexto:
    - Después de recuperar la información relevante, se ajusta el contexto para proporcionárselo al modelo generativo. El contexto incluye:
        - Estado del Juego: Una descripción del rol y propósito del chatbot especializado en *The White Castle*.
        - Pregunta del Usuario: La consulta original del usuario.
        - Información Recuperada: Los fragmentos relevantes obtenidos en la fase anterior de recuperación.
7. Generación de Respuesta con un Modelo Generativo:
    - Se utiliza un modelo de lenguaje preentrenado, Zephyr, para generar una respuesta basada en el contexto ajustado. El modelo genera respuestas coherentes y precisas basadas en la información recuperada y en el contexto general del juego.
    - La generación es controlada mediante parámetros como el número máximo de tokens, temperatura (para la diversidad de las respuestas) y otros, lo que permite un control sobre la longitud y estilo de la respuesta.
8. Posprocesamiento de la Respuesta:

    - Si la respuesta se genera en inglés pero el usuario originalmente hizo la consulta en español, la respuesta generada se traduce automáticamente al español utilizando un traductor.
9. Presentación de la Respuesta:
    - Finalmente, la respuesta generada (ya sea en el idioma original o traducida) se presenta al usuario, proporcionándole una solución clara y directa a su consulta.

## Agente Inteligente con ReAct

### Funciones de Búsqueda:

- Existen tres funciones principales de búsqueda, cada una destinada a interactuar con diferentes tipos de bases de datos:
    - doc_search(query):
        - Realiza una búsqueda en la base de datos vectorial utilizando la clase DocSearch.
    - graph_search(query):
        - Esta función consulta la base de datos gráfica (Neo4j) mediante la clase GraphSearch.
    - table_search(query):
        - Se conecta con la base de datos tabular (almacenada en un DataFrame de Pandas, df_castle) y realiza una búsqueda en la tabla con datos estructurados como calificaciones, número de jugadores, precio, etc.

### Herramientas del Agente:

- Se definen las herramientas para el agente utilizando FunctionTool.from_defaults:
    - graph_search: Busca información en la base de datos de grafos.
    - table_search: Busca información en la base de datos tabular.
    - doc_search: Busca información en la base de datos vectorial.

Estas herramientas permiten que el agente ejecute consultas específicas en diferentes bases de datos según el tipo de información solicitada.

### Configuración del LLM:

- Se configura un modelo LLM (Large Language Model) llamado Llama 3.2 a través de la clase Ollama. Este modelo es el que utiliza el agente para generar respuestas y realizar razonamientos basados en las consultas del usuario.
- Los parámetros clave incluyen:
    - request_timeout: Define el tiempo de espera para las solicitudes.
    - temperature: Controla la creatividad en las respuestas generadas.
    - context_window: Controla el tamaño del contexto utilizado por el modelo.

### ReActAgent:

- Se crea el agente ReAct utilizando la clase ReActAgent.from_tools, pasando las herramientas definidas anteriormente y el modelo LLM. El agente tiene un conjunto de instrucciones para guiar el razonamiento y la acción:
    - system_prompt: Define el papel del agente y las reglas que debe seguir al interactuar con el usuario, como no inventar información y solo utilizar datos obtenidos de las herramientas disponibles.
    - chat_formatter: Formatea las interacciones del agente con el usuario.
    - react_chat_history: Desactiva el historial de chats, lo que significa que el agente no mantiene un registro de las interacciones pasadas.

### Función de Interacción con el Agente:

- chat_with_agent(query):
    - Esta función maneja las consultas del usuario. Detecta si la consulta está en español y la traduce al inglés si es necesario, luego la pasa al agente ReAct para obtener una respuesta.
    - Si la consulta original es en español, la respuesta del agente se traduce de vuelta al español.
    - Si hay un error durante el procesamiento de la consulta, se registra un mensaje de error.

## Herramientas y Tecnologías Empleadas

- **Lenguaje de Programación**: Python fue el lenguaje elegido.
- **Entorno de trabajo:** Se desarrolló todo el proyecto en un cuaderno de Google Colab.
- **Librerías Utilizadas**: Las librerías que se usaron están detalladas en el cuaderno mencionado anteriormente ya que mencionarlas aquí no sería adecuado debido a su

cantidad, las principales involucran procesos de PLN como Llama-Index para el agente, Hugging Face para los pipelines de modelos y Selenium junto a BeautifulSoup para el scrapping.

- **Bases de Datos**: Se empleó pandas para los datos **tabulares,** Neo4j para datos estructurados en grafos y ChromaDB para los embeddings los cuales fueron almacenados en una base **vectorial** para optimizar la búsqueda híbrida.

## Pasos Seguidos para Cumplir los Objetivos

1. Recolección de datos relevantes mediante técnicas de web scraping.
2. Creación de las bases de datos para organizar la información de manera eficiente.
3. Implementación de los métodos de búsqueda híbrida para la recuperación de información en base de datos vectorial.
4. Implementación de dos métodos dinámicos para la recuperación de información en base de datos de grafos y tabular.
5. Desarrollo y comparación de clasificadores.
6. Integración completa de RAG.
7. Desarrollo del agente inteligente con capacidades de razonamiento basadas en ReAct.
8. Evaluación del sistema en el contexto del juego.

# Resultados

## Resultados del Sistema RAG

### Consultas sobre los Diseñadores y Artistas:

- Consulta: "Who designed this game?"
- Respuesta generada: *The designers of The White Castle are Isra and Shei.*
- Consulta: "Who covered the art of the game?"
- Respuesta generada: *The art of The White Castle is covered by an artist named Kwanchai Moriuchi.*

Las respuestas obtenidas fueron precisas y específicas en cuanto a los diseñadores y el artista que cubrió el arte del juego. El modelo identificó correctamente la información relevante en la base de datos de grafos, destacando relaciones de entidades clave asociadas con el juego.

### Consulta sobre Otros Artistas:

- Consulta: "Is there any other artist behind the game?"
- Respuesta generada: *El artista detrás de El Castillo Blanco es Joan Guardiet.*

El sistema pudo generar una respuesta adecuada en español, aunque en algunos casos, podría beneficiarse de una mayor validación contextual para confirmar si el nombre corresponde a otro artista o si se trata de un error en la base de datos.

### Consultas sobre el Precio y la Clasificación:

- Consulta: "What is the rating of this game?"
- Respuesta generada: *The rating of this game is 8.0.*
- Consulta: "How much does it cost?"
- Respuesta generada: *El precio de The White Castle no se especifica en la información proporcionada.*

El sistema maneja bien las consultas relacionadas con datos tabulares, como la clasificación y el precio, aunque en el caso del precio, el modelo no pudo obtener una respuesta válida debido a un error en la consulta (Price no está definido). Este es un punto de mejora, ya que la consulta debería poder recuperar correctamente el valor asociado con el precio.

### Consultas sobre Reglas y Consejos de Juego:

- Consulta: "Could you give me the basic rules to start playing?"
- Respuesta generada: *Para jugar a The White Castle, cada jugador comienza con un personaje y un conjunto de recursos. Tu objetivo es llegar al Castillo Blanco...*
- Consulta: "What rules or mechanics do I need to improve to become a better player?"
- Respuesta generada: *To enhance your gameplay in The White Castle, you may want to focus on mastering the combat system...*

El modelo fue capaz de proporcionar reglas básicas y consejos para mejorar en el juego. Esto demuestra que el sistema puede generar respuestas útiles y contextualmente correctas para jugadores con diferentes niveles de experiencia, mejorando la interacción general del usuario con el sistema.

### Consultas sobre Colores de las Figuras:

- Consulta: "What are the available colors for the pawns?"

- Respuesta generada: *La lista de colores disponibles para las figuritas es la siguiente: Rojo, Azul, Verde, Amarillo, Negro.*

Esta consulta fue completamente procesada y el sistema devolvió la lista esperada sin problemas, demostrando un buen manejo de los datos de categorías específicas dentro del juego.

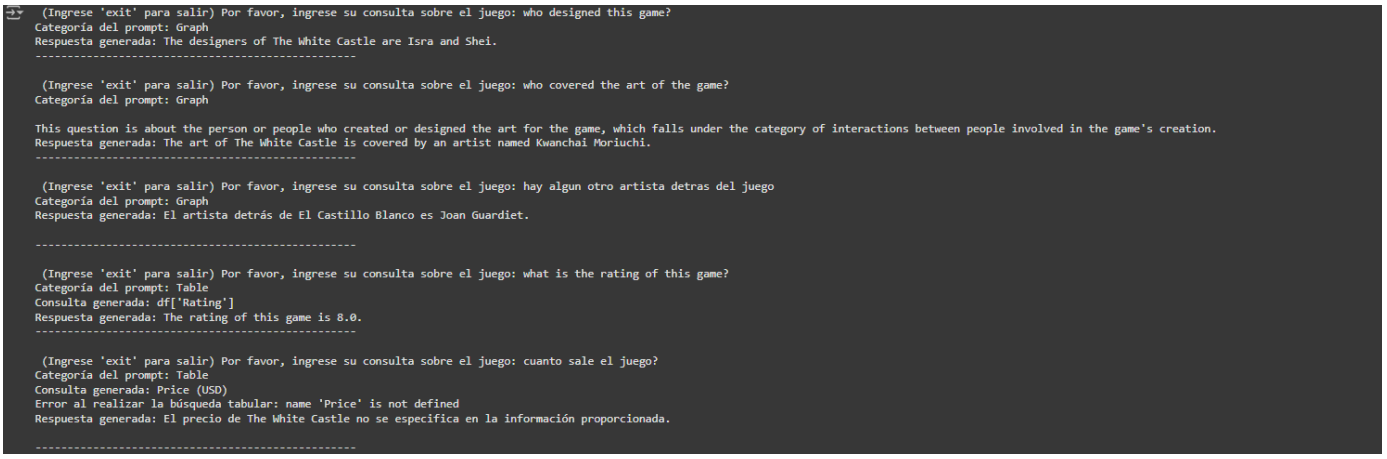



Imagen capturada sobre los resultados de algunas de las preguntas dadas y como el sistema fue capaz de responder bien y en ambos idiomas

# Resultados del Agente

## Query 1: Who designed the game and what's the minimum number of players to play it?

- Problema: El agente no fue capaz de obtener una respuesta válida y se encontró con un error relacionado con "Reached max iterations".
- Posible causa: La consulta requiere acceder tanto a la base de datos gráfica (para el diseñador) como a la base de datos tabular (para el número mínimo de jugadores). Podría estar ocurriendo una sobrecarga o mal manejo en la integración de las herramientas.
- Resultado esperado: El agente debería combinar la información de ambas fuentes correctamente.

## What are the game rules and what's the rating of the game?

- Resultado: El agente devuelve correctamente el rating del juego, pero no menciona las reglas.

- Posible causa: El agente está buscando solo en la base de datos de documentos, pero no ha proporcionado una respuesta completa sobre las reglas. Esto podría deberse a que la base de datos no contiene una descripción suficiente de las reglas o el sistema no está estructurado correctamente para devolver toda la información.
- Resultado esperado: Debería devolver tanto las reglas como el rating de forma completa.

### Query 3: How complex is the game and how many likes from people does it have?

- Resultado: El agente da una respuesta precisa sobre la complejidad del juego, pero no menciona el número de "likes".
- Posible causa: La base de datos tabular probablemente contiene información sobre la complejidad y los likes asique se podria deber a un mal funcionamiento en TabularSearch.
- Resultado esperado: El agente debería proporcionar tanto la complejidad como la cantidad de "likes" sin omisiones.

### Query 4: Who covered the art of the game and how much time does it take?

- Problema: El agente no puede acceder a la columna de 'Artist' en el dataframe y genera un error en la búsqueda.
- Posible causa: El sistema tiene un error relacionado con el acceso a columnas que no existen en el dataframe, lo que provoca un fallo en la búsqueda. Además, el agente no pudo manejar el error adecuadamente.
- Resultado esperado: El agente debería poder acceder a la información sobre el arte y el tiempo de juego sin errores.

### Query 5: ¿En qué año se lanzó el juego y cuántos jugadores puede tener como máximo?

- Problema: El modelo no decidió correctamente a donde ir a buscar la información lo que causó que se busque información que no está contenida en la base de datos de grafos.

- Resultado esperado: El agente debería devolver el año de lanzamiento y el número máximo de jugadores sin generar errores.

---

### Ejemplos donde el agente falla o las respuestas no son precisas

1. Query 2 (Reglas del juego y rating):
   - Problema: Aunque el agente menciona el rating, no proporciona información suficiente sobre las reglas del juego. Esto sugiere que la base de datos de documentos no tiene los detalles necesarios o que el agente no consulta de forma efectiva.
2. Query 4 (Arte y tiempo de juego):
   - Problema: El agente no pudo acceder a la información sobre el arte, lo que podría indicar que la columna Artist no está correctamente configurada en el dataframe o que el agente no maneja bien las consultas con errores.
3. Query 5 (Año de lanzamiento y jugadores):
   - Problema: El sistema de Neo4j muestra warnings relacionados con propiedades y relaciones desconocidas, lo que impide que el agente devuelva los resultados esperados. Este error puede indicar que el grafo no está correctamente estructurado o que las relaciones no se están manejando adecuadamente.

---

## Mejoras recomendadas

### Manejo de errores en las herramientas:

1. Optimización en la integración de herramientas:
2. Manejo de errores en las herramientas
3. Revisar y mejorar los métodos de búsqueda,
4. Mejoras en la precisión de las respuestas

# Conclusiones

## RAG

Precisión y relevancia: Las consultas más relacionadas con las relaciones en la base de datos de grafos (diseñadores, artistas) y las reglas del juego fueron bien manejadas. El sistema proporcionó respuestas detalladas y relevantes para el usuario, mejorando la interacción.

Limitaciones: Los errores en la consulta sobre el precio del juego destacan la necesidad de ajustes en el sistema para mejorar el manejo de consultas tabulares y datos no definidos. Este tipo de consulta podría ser un área crítica para mejorar la robustez del sistema. Además se sabe que tanto la búsqueda en grafo como la búsqueda en tabla trabajan con base de la misma idea de usar un modelo de apoyo

Potencial de mejora: Aunque las respuestas generadas son correctas en términos generales, el sistema podría beneficiarse de una validación más avanzada y una mejor integración de la información de la base de datos, especialmente para consultas de datos no estructurados o inconsistentes.

## ReAct Agent

El modelo ReAct, aunque promete una integración eficiente entre la generación de respuestas y la búsqueda de información en diversas fuentes (como bases de datos gráficas, tabulares y de documentos), presenta varios problemas de integración que afectan su rendimiento. En los ejemplos evaluados, se observó que el agente no logra coordinar de manera fluida las búsquedas entre diferentes herramientas, lo que lleva a errores y respuestas incompletas.

### Algunos de los problemas clave que se detectaron incluyen:

1. Inconsistencias en la búsqueda y extracción de datos: El agente no siempre logra acceder a las propiedades y relaciones correctas dentro del grafo de Neo4j, lo que provoca errores como "UnknownPropertyKeyWarning". Esto sugiere que la integración con la base de datos no es lo suficientemente robusta y las consultas no están correctamente ajustadas a la estructura del grafo.
2. Manejo inadecuado de errores: El agente no maneja adecuadamente las excepciones generadas por las búsquedas fallidas. Cuando no se puede encontrar una propiedad o columna esperada en el dataframe o en la base de datos, el proceso se interrumpe sin

proporcionar una retroalimentación clara, lo que limita la capacidad del sistema para seguir procesando la consulta de manera efectiva.

3. Falta de combinación efectiva de resultados: En consultas que requieren acceder a más de una fuente de datos (como el diseñador del juego y el número mínimo de jugadores), el agente no logra combinar correctamente los resultados de las distintas herramientas, lo que lleva a respuestas parciales o erróneas.

En resumen, aunque el modelo ReAct tiene el potencial de ser una herramienta poderosa para combinar generación de texto y búsqueda de información, la integración de sus componentes aún necesita mejoras significativas. Para lograr una interacción fluida y precisa, se debe optimizar el manejo de consultas, la estructuración de los datos en el grafo y la base de datos tabular, así como el sistema de manejo de errores. Solo entonces el modelo podrá ofrecer respuestas más completas y coherentes en escenarios de búsqueda múltiple.

# Conclusión General

Este trabajo ha abordado el desarrollo e implementación de un sistema de Recuperación de Información Aumentada (RAG) utilizando un agente inteligente con el enfoque ReAct para responder consultas complejas sobre el juego "The White Castle". El sistema se diseñó con el objetivo de combinar múltiples fuentes de datos, incluyendo bases de datos Neo4j, bases de datos tabulares y bases de datos vectoriales, para proporcionar respuestas precisas y contextualizadas.

## Principales Resultados:

- Implementación exitosa de bases de datos: La creación de las bases de datos vectorial y de grafos permite organizar los datos de manera coherente y accesible, mejorando la eficiencia en las consultas.
- Desarrollo de métodos de búsqueda: Se implementaron diversos métodos de búsqueda (DocSearch, TabularSearch, GraphSearch) para cubrir diferentes tipos de consultas. Estos métodos permitieron acceder a la información relevante de manera rápida y estructurada.
- Clasificadores y uso de modelos LLM: Se desarrollaron clasificadores tanto basados en ejemplos como en modelos de lenguaje preentrenados (LLM), lo que permitió una clasificación precisa de las consultas y la integración de respuestas de diferentes fuentes.
- Resultados del Agente ReAct: El sistema ReAct, en conjunto con las herramientas desarrolladas, ofreció un enfoque prometedor para generar respuestas a consultas complejas. Sin embargo, se identificaron bastantes problemas, como la integración imperfecta de herramientas, errores de consulta, y falta de robustez en el manejo de excepciones.

## Áreas de Mejora:

- Optimización en la integración de herramientas: La falta de una integración fluida entre las búsquedas gráficas, tabulares y vectoriales fue uno de las principales

falencias. Mejorar la cohesión entre estos componentes permitirá al sistema generar respuestas más completas y precisas.

- Manejo de errores: A pesar de los esfuerzos por manejar las excepciones, se observaron fallos al procesar ciertas consultas. Es necesario fortalecer el manejo de errores para evitar que el sistema se detenga ante consultas inesperadas o problemas con los datos.
- Mejoras en la recuperación de información aumentada: Se podría optimizar la manera en que el agente utiliza las herramientas para recuperar información de diversas fuentes, asegurando una mayor precisión en la combinación de los resultados.

## Reflexión Final:

En resumen, este trabajo ha logrado, en su primera parte, construir un sistema avanzado para responder preguntas complejas utilizando la recuperación de información aumentada RAG.

y en su segunda etapa, a pesar de algunos desafíos técnicos y errores de integración, el enfoque ReAct mostró un gran potencial en la capacidad del agente para generar respuestas contextuales. Las mejoras en la integración de herramientas, el manejo de errores y la optimización de la recuperación de información serán clave para potenciar la efectividad y la utilidad de este sistema en el futuro.