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

Procesamiento de Lenguaje Natural
Trabajo Práctico Final

18/12/2024

Pace, Bruno E. P-5295/7.

Docentes: Manson Juan Pablo, Geary Alan, Ferrucci Costantino, Sollberger Dolores.

Contenidos

1 Introduccion

En este informe se desarrolla el Trabajo Práctico Final de la materia "Procesamiento de Lenguaje Natural" de la carrera Tecnicatura universitaria en Inteligencia Artificial de la Facultad de Ciencias Exactas, Ingeniería y Agrimensura de la Universidad Nacional de Rosario.

El Trabajo consta de dos ejercicios: el primero con la generación de un RAG de un juego Eurogame, *Rajas of the Ganges*, capaz de interpretar y comunicarse en inglés o español. El segundo ejercicio, trata de la implementación de un agente basado en ReAct.

El desarrollo y estrategias empleadas en este proyecto, plasma la eficiencia de los conocimientos aprendidos durante la cátedra. Entre los recursos encontramos: Web Scrapping, confección de Bases de Datos (tabulares, grafos, vectoriales), modelos de embeddings (representación vectorial de palabras), herramientas de lectura de PDF, análisis semántico, análisis de entidades, LLM's para distintos usos y un largo etcétera.

Todo el trabajo está confeccionado en el lenguaje Python.

1.1 Rajas Of The Ganges

Rajas of the Ganges es un juego de mesa estratégico ambientado en la era dorada del Imperio Mogol en la India. En una una época marcada por la expansión, el comercio próspero de productos como seda, té y especias, y la construcción de monumentos emblemáticos como el Taj Mahal y el Fuerte Rojo. Los jugadores asumen los roles de rajas y ranis (líderes y soberanos) que buscan expandir sus dominios, mejorar sus propiedades y ganar influencia en este vasto imperio.

El objetivo principal del juego es encontrar el equilibrio entre dos recursos clave: el prestigio y la prosperidad, ya que ambos están ligados al concepto del Karma en el juego. A través de decisiones estratégicas, los jugadores competirán para construir y desarrollar sus provincias, optimizando sus acciones para alcanzar el mayor éxito en ambos aspectos. Al hacerlo, tratarán de destacar como el líder más influyente y próspero del imperio, asegurando así su lugar en la historia.

2 Abordaje

El problema se aborda en inglés, ya que los modelos utilizados no son de licencias de pago y no ofrecen un rendimiento excepcional. Para maximizar su eficiencia, todas las bases de datos están en este idioma. Por lo tanto, si el prompt está en español, se traduce al inglés. De igual manera, para las salidas en español, los resultados se traducen desde el inglés.

3 Enfoque

El trabajo sigue buenas prácticas de programación, desde un paradigma funcional. No sólo por prolijidad, sino por escalabilidad. Las funciones son de fácil interpretabilidad y mediante ellas se logra modularización. Estas funciones están bien documentadas, con datos tipados y comentarios donde corresponde. Esto facilita cambios durante el proceso de desarrollo e incluso posibles cambios que puedan surgir en producción.

4 Problema nro. 1: RAG

4.1 Resumen

Previo a la creación del RAG, fue necesario extraer la información necesaria para crear las bases de datos(tabular, de grafos, vectorial). Luego, se crearon clasificadores para el prompt, uno con una Regresión Logística y otro con un LLM. Para mejorar la eficiencia del programa, se incorporan consultas dinámicas para la base de datos tabular y de grafos. Para la base de datos vectorial, se obtiene la mejor coincidencia para el documento mediante embeddings y por comparación de palabras. Finalmente, se crea el RAG, restringiendo que no responda preguntas que no estén contempladas en la base de datos. Podemos adelantar que el rendimiento obtenido es bueno. El modelo logra responder las preguntas que el usuario le hace.

4.2 Desarrollo

Tenemos tres distintas bases de datos: grafo, tabular y vectorial.

4.2.1 Base de datos de Grafo

Para la base de datos de grafo, se utilizó el motor *Redis*. Este es fácilmente utilizable e integrable con Google Colab. También, soporta *Cypher*. Cypher goza de una sintaxis simple, lo cual facilitó la interpretabilidad de las consultas.

Los datos de esta base de datos fueron extraídos de *BGG - Rajas Of The Ganges: créditos* mediante Web Scrapping con *Selenium*. Esto se debe a que BGG tiene secciones que utilizan JavaScript, lo cual dificultaba la tarea a librerías como BeautifulSoup. Selenium controla en este caso un navegador web Google Chrome, que tiene un timer para esperar que los datos carguen en la página antes de extraerlos, evitando errores.

La base de datos de grafo, tiene como centro un nodo con el título del juego, al cual están todos los demás nodos conectados. Contiene relaciones de: categorias del juego, juegos similares, publicadores, mecánicas del juego, sus diseñadores y artistas creadores. Para ver el gráfico del grafo con más detalle *click aquí*.

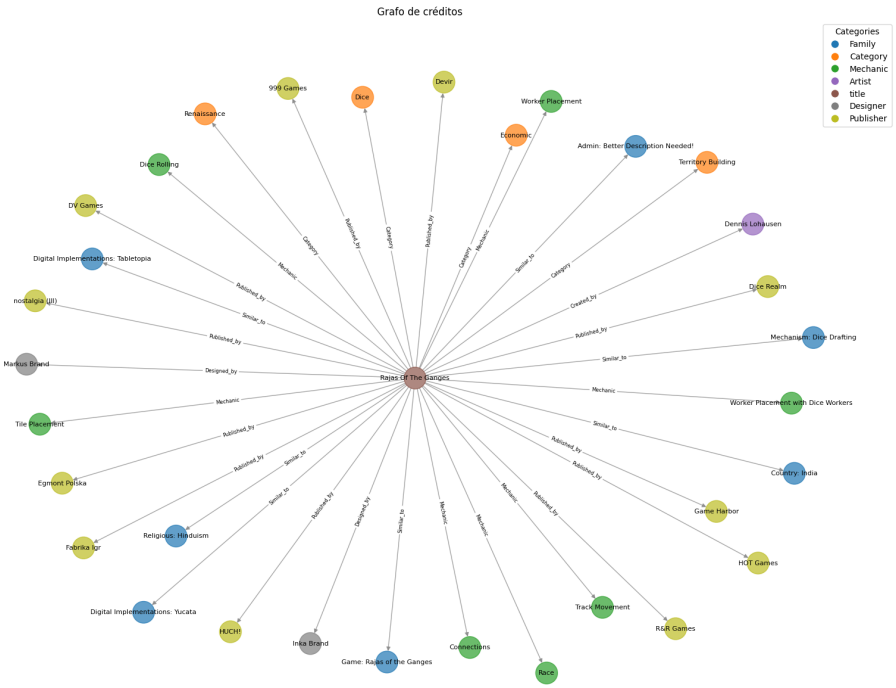

Figure 1: Grafo de créditos del juego.

4.2.2 Base de datos tabular

Para la base de datos tabular se emplea un DataFrame de *Pandas*. Esto es eficiente y simple ya que es una herramienta sumamente conocida.

Los datos del DataFrame son:

- Promedio de valoraciones.

- Cantidad de valoraciones.
- Desviación estándar de valoraciones.
- Valoraciones.
- Comentarios.
- Fans.
- Cantidad de visitas a la página.
- Ranking general.
- Ranking de estrategias.
- Todas las partidas del juego.
- Jugadores este mes.
- Personas que tienen el juego.
- Personas que compraron el juego, pero ya no lo tienen.
- Piezas para cambiar.
- Piezas que quieren para cambiar.
- Lista de deseos (el juego está en la lista de deseos de personas).
- Personas que tienen piezas para intercambiar.
- Personas que quieren piezas para intercambiar.

Estos datos fueron extraídos de *BGG - Rajas of The Ganges: stats*. La extracción se realizó mediante Web Scrapping con *Selenium* por lo descrito en la sección anterior (2.2.2).

4.2.3 Base de datos Vectorial

Para esta base de datos se utiliza *ChromaDB*. Esta tiene datos de distintas fuentes.

Documentos de texto

Mediante un research de documentos del juego, se encontraron en la página *BGG - Rajas of The Ganges: files* tres PDF:

- Reglas del juego.
- Reglas de la versión automa.
- Explicación simple.

También, se extrajo desde la página de Board Gamer Review una review del juego.

A estos documentos, se les extrae el texto con una simple línea de código con la libreria PyPDF2. Este texto es almacenado en variables que más tarde son partidas en chunks.

En todos los casos, los archivos están almacenados en git y se los obtiene de forma automática desde el notebook.

Nota: el documento con los datos de la versión automa, fue comentado en el código porque afectaba el rendimiento del modelo. Esto se amplía en una sección posterior.

Videos de Youtube

Otra fuente de datos son los videos de youtube. Los videos utilizados son:

- ¿Cómo se juega?
- Review
- Gameplay

Se les extrae el texto mediante youtube-transcript-api. Esta herramienta transcribe a texto el audio de los videos mediante un ID. La función implementada en python encargada de hacer esta tarea, toma un URL y extrae el ID, para ser más eficiente. Estas transcripciones se guardan en variables.

En ambos casos, tanto con los documentos como las transcripciones de los videos, se hace una limpieza de saltos de línea.

Chunks

Continuando con el procesado de los documentos de texto y videos, se partieron en chunks. El proceso consiste en separar los cuerpos de texto extensos en porciones más chicas. En este caso, se utilizó un tamaño de 500 caracteres, con superposición de 50. Esta superposición marca cuantos caracteres se pueden tener repetidos en un chunk, de su chunk anterior. Sirve para no cortar la idea de una frase.

Para realizar esta tarea, se utilizó por recomendación de la cátedra RecursiveCharacterTextSplitter del framework LangChain.

Embeddings

Para convertir los chunks (fragmentos de texto) en vectores numéricos, se utilizó la herramienta SentenceTransformer. Esta elección fue el resultado de un proceso de prueba y error. Durante la fase de selección, se evaluaron diferentes modelos, incluyendo una variante de BERT (multicased), que fue descartada debido a que generaba distancias excesivamente largas en las respuestas a las consultas realizadas sobre la base de datos, simulando un entorno funcional del RAG. Asimismo, se probó el modelo intfloat/multilingual-e5-large, que también fue descartado por su elevado consumo de recursos en Google Colab, lo que provocaba un colapso del sistema y hacía inviable su implementación. En internet, se encontró el modelo embed-english-v3.0 pero tampoco tuvo éxito en el funcionamiento en colab.

Nota: en el notebook, encontramos dos funciones de embedding. La comentada denota una parte del proceso comentado de selección mediante pruebas y errores.

Teniendo así los embedding listos, se almacenan en la Chromadb con un id único generado con uuid, el chunk (texto) y su correspondiente embedding.

4.2.4 Clasificadores

El clasificador es el que se encarga generar una etiqueta para el prompt del usuario. Este puede tener las siguientes etiquetas:

- Rules.
- Review.
- Gameplay.
- Stats.
- Graph.

La etiqueta es la que determina en qué base de datos buscar al momento de obtener información en el retriever. Si la etiqueta es "graph", busca en el grafo, si es "stats" busca en el DataFrame y en caso que sea "rules", "review" o "gameplay" busca en la base de datos vectorial.

Se crean tres clasificadores: dos basados en ejemplos y embeddings y otro generado con un modelo *Qwen/Qwen2.5-72B-Instruct*.

Modelo basado en ejemplos y embeddings

Primeramente, se crea un DataFrame que contiene como dato un prompt y una etiqueta. Este es un dataset sintético y simula posibles prompts que un usuario pueda darle al RAG.

Con fines comparativos, se crean dos clasificadores de embeddings y ejemplos. Uno con embeddings mediante un transformer y otro con TF-IDF.

Regresión Logística: TF-IDF

Se utiliza en esta ocasión TfidfVectorizer. Se hace el entrenamiento ($fit\_transform$) con los datos de train para no sesgar los resultados. Por eso al conjunto de test, se le aplica solo un ($transform$).

TfidfVectorizer es capaz de medir la relavacia de las palabras en los documentos con relción al resto del texto, asignándole más peso a las que aparecen menos veces. Se basa únicamente en la frecuencia de los términos, sin capturar contexto ni semántica. También, genera vectores de alta dimensionalidad, en los que cada palabra representa una dimensión.

Mediante el framework scikit-learn se utiliza *train_test_split*, destinando un 80% al entrenamiento y un 20% para test. Se utiliza un modelo de LogisticRegression que se adapta bien a los problemas multiclases.

La métrica para comparar es F1-score. F1-score al tener en cuenta el recall y la precision, es el indicado para comparar ambos modelos. Queremos diferenciar bien las clases, lo que nos llevará a un mejor rendimiento del RAG.

F1-score (macro) : 0.70.

Regresión Logística: embeddings con transformers

Se aplica una LogisticRegression muy similar que en el modelo anterior. Varía la forma de codificar las frases. En este caso se utiliza un transformer para generar embedding, con el mismo modelo mencionado en la seccion 4.2.3. Este modelo diseñado para generar representaciones de texto en un espacio de alta dimensionalidad es capaz de distinguir relaciones semánticas entre palabras (incluso si no son iguales) y contexto de palabras dentro de una frase. Genera embeddings de valores flotantes en todas las dimensiones.

Obtiene un F1-Score (macro) de 0.76.

Diferencias entre ambas Regresiones Logísticas

Entre las principales diferencias encontramos:

- Semántica y contexto: el transformer es capaz de capturar semántica y contexto de la frase, teniendo en cuenta palabras similares como sinónimos. En cambio, TF-IDF no es capaz, ya que se basa solo en la frecuencia de aparición.
- Dimensionalidad: TF-IDF tiene una dimensión variable, que se basa en el tamaño del vocabulario. En cambio, el transformer tiene como resultado embeddings de tamaños fijos.
- Generalización: el transformer es capaz de adaptarse mejor frente a palabras nuevas ya que son entrenados con grandes conjuntos de datos. En cambio, TF-IDF tiene dificultades para enfrentar palabras que no figuran en el corpus de texto original.

Estas diferencias, ilustran por qué el transformer (modelo complejo) es mejor, obteniendo una métrica superior a TF-IDF (modelo más simple). Aunque existen estas diferencias, el modelo más complejo destaca no demasiado.

LLM

Una vez más, el modelo elegido es *Qwen/Qwen2.5-72B-Instruct*. Se le explicitan las etiquetas posibles y algunos ejemplos de prompts con la etiqueta deseada.

Se realiza una prueba con el mismo dataset con el que se testeo el modelo con LogisticRegression, obteniendo un F1-score (macro) : 0.92.

Modelo clasificador elegido: LLM

Teniendo en cuenta que los modelos mediante LogisticRegression (TF-IDF) tiene un F1-score (macro) de 0.70, el de LogisticRegression (transformer) obtiene 0.76 y el LLM de 0.92, se opta por LLM. Esto nos llevará a un rendimiento robusto y superior del clasificador y por ende, del LLM en general.

4.2.5 Consultas Dinámicas

Para la consulta dinámica del grafo, se emplea un modelo *Qwen/Qwen2.5-72B-Instruct*. Este modelo es capaz, mediante instrucciones, generar una query para obtener sólo la data necesaria del grafo y en base a eso, posteriormente generar una respuesta. Se le informa al modelo qué datos tiene almacenado el grafo en su interior para esto y ejemplos de la salida esperada.

Listing 1: Ejemplo de query dinámica de grafo

```
get_graph_query('what␣are␣the␣mechanics␣of␣the␣game?', graph_data)
Return:
MATCH (title:title {name: "Rajas␣Of␣The␣Ganges"})-[:Mechanic]->(
    mechanic:Mechanic) RETURN mechanic

graph_retriever('what␣are␣the␣categories␣of␣the␣game?')
Return:
['Dice', 'Economic', 'Renaissance', 'Territory␣Building']
```

Luego, encontramos el retriever que sólo

En el caso de la base de datos tabular (para esta situación, un DataFrame), se utilizó el mismo modelo *Qwen/Qwen2.5-72B-Instruct*. Siendo este capaz de generar un filtro de *Pandas* para extraer solo la información inherente a la pregunta (prompt) del usuario. De la misma manera que en el caso de la consulta dinámica de grafos, se le da la información del DataFrame y la salida esperada con ejemplos.

Listing 2: Ejemplo de query dinámica de DataFrame

```
get_df_query('How␣many␣players␣played␣this␣month?', df_stats_long)
Return:
df_stats_long.loc[df_stats_long['Stat'] == 'this_month', 'Value']
```

A posteriori, encontramos el retriever del DataFrame que no tiene otra función más que dado prompt, devolver el dato necesario.

4.2.6 Busqueda en base de datos Vectorial

Similitud Semántica

Para la búsqueda mediante similitud semántica, se obtiene el embedding del prompt de la misma forma que se obtuvieron los de toda la base de datos y se busca mediante una query de chroma, en la cual devuele los documentos y distancias al prompt de los primeros 10 resultados. Las distancias son devueltas para luego usarlas en el re-rank.

Búsqueda por palabras claves

Para buscar por palabras claves, primero se aplica una operación sobre los documentos de la base de datos vectorial. Esta consta en: pasarlos a minúsculas, separarlos en palabras y eliminar los caracteres que no son letras. De esta manera eliminamos información que no es relevante. Con este fin, se utiliza $NLTK$, una herramienta de lenguaje natural. Puntualmente *word_tokenize* es el encargado de partir en palabras una frase más larga. Estas mismas operaciones se le aplican al prompt.

Luego, con rank-bm25 en su variante BM25Okapi. Este modelo calcula un puntaje de las palabras clave en base a su frecuencia de aparición. Entonces se genera la búsqueda por palabras y devuelve los 10 mejores documentos.

4.2.7 Rerank

Para el rerank se encuentra la dificultad de las distintas "escalas" en las que mide cada algoritmo. Por un lado, tenemos distancias de la query de chroma y por otro el score de

BM25. Para resolver esto y poder compararlas, se aplica un estandarizado min-max y una multiplicación por dos. De esta manera, la nueva escala es de [0,2] y son comparables.

Finalmente, se crea un retriever de la base de datos vectorial. Este se encarga de devolver los documentos rankeados, llamando a las funciones de rerank, busqueda semantica y busqueda por palabras claves.

4.2.8 Interacción con el usuario

Dentro de la interacción con el usuario, tenemos dos principales secciones: una que se encarga del manejo de los idiomas (inglés y español) y otra dedicada exclusivamente a la creación del modelo de rag.

Traducción de texto

De forma interna, todas las bases de datos y estructuras de datos se trabajan en inglés. Para poder manejar los dos idiomas en una misma ejecución del modelo RAG, tenemos que tener dos pilares fundamentales: un detector de idioma y un traductor.

Para el detector de idiomas, utilizamos Langdetect. Mediante su metodo (*detect*), podemos saber si el usuario está interactuando en inglés o en español.

También, tenemos un traductor que utiliza la librería de python translator. Se encarga de traducir de español a inglés y viceversa. Esto lo hace para poder tomar un prompt en español, traducirlo al inglés, hacer la búsqueda y con la respuesta elaborada en inglés, traducirla nuevamente a español.

De esta forma, interpreta ambos idiomas.

Creacción del RAG

En esta instancia, para simplificar el código, tenemos un (*retriever_caller*) que se encarga de llamar al retriever según corresponda con la etiqueta y devolver los documentos o respuestas asociadas mediante el prompt.

Para crear el modelo, utilizamos *Qwen/Qwen2.5-72B-Instruct*, explicándole que es un RAG del juego *Rajas of The Ganges* y que no debe responder preguntas de las cuales no tenga información, devolviendo que no cuenta con la información necesaria en ese caso.

Nota: se dejó una lista de frases con un bucle for para que el notebook corra de forma automática, pero la función de interacción con el usuario existe.

Aquí vemos un ejemplo de rendimiento:

```
Usuario:  quien es el creador?
Chatbot: El creador de Rajas of the Ganges es Dennis Lohausen.


Usuario:  best estrategy
Chatbot: The best strategy in Rajas of the Ganges involves efficiently
   placing your workers to maximize Fame points, particularly by
   advancing your temple tracks and securing high-value actions. After
   a few rounds, the game mechanics become more intuitive, allowing you
    to focus on advanced strategies and optimal moves.

Usuario:  quien publico el juego?
```

```
Chatbot: El juego fue publicado por 999 Games, Devir, Dice Realm, DV
    Games, Egmont Polska, Fabrika Igr, Game Harbor, HOT Games, nostalgia
     (III), y R&R Games.

Usuario:  What do people say about the game?
Chatbot: People say that Rajas of the Ganges is a game that is very
    mechanical in some senses but is fun due to its various engaging
    elements such as collecting and rolling dice, building up tiles, and
     worker placement. They also mention that the game has a fast-paced
    nature and an interesting method of victory involving two victory
    point tracks that need to cross. Some players enjoy the standard
    version of the game more, finding it easier to understand and enjoy,
     while others appreciate the variants, such as the river tiles,
    which add variety and change the game dynamics. Replayability may
    not be a strong point, but the game remains enjoyable even after
    multiple plays.
```

5 Desafíos

Se pueden destacar distintos desafíos que se han enfrentado durante el desarrollo del RAG. Entre ellos:

- Rendimiento de la librería LangDetect. En ocasiones, se detectaban idiomas erróneos. Por ejemplo, cuando se introduce la frase "similar games", devuelve un "sq". Esto complicó la correcta detección y decantó en una estrategia que tiene en cuenta esos casos. En otras palabras, ante la duda devuelve "en" (inglés).
- GPU en colab: Limitaciones para la elección de modelos. Colapsa con el modelo mencionado e5-multilingual. Es una de las razones por las que se desarrolla todo el trabajo en inglés. Además, se agotaba la cuota de uso con la T4, generando una demora considerable en el tiempo de ejecución para debuggear el código.
- Al agregar información al RAG, se notaba una merma en el rendimiento. Luego de investigar por qué, se definió que el documento que contenía las reglas 'automa' generaba errores en las respuestas del modelo. Por ende, se deja constancia del trabajo de research, pero no se utiliza.

6 Conclusiones

Se consiguió una versión del RAG muy buena, que puede responder con certeza las preguntas planteadas, como podemos ver en el ejemplo. Las consultas suelen ser muy eficientes. Es una herramienta eficiente incluso con grandes conjuntos de datos de texto como tiene la base de datos vectorial. Además, todo el ejercicio se presenta en automatizado, evitando al máximo el "hardcoding".

7 Problema nro. 2: Agente

7.1 Resumen

Con la base del RAG, se emplea un agente ReAct con sus respectivas herramientas . Este se encarga de tomar las decisiones entre el prompt y las bases de datos. Se puede adelantar que alucina demasiado y que varía sus respuestas dependiendo la ejecución.

7.2 Desarrollo

El desarrollo del agente en ReAct, incluye la utilización de otro modelo distinto al del resto del trabajo. Este modelo de Ollama llamado Phi-3 : Medium tiene 14 Billones de parámetros. Comparando con el qwen de 72 Billones que se utilizó en el ejercicio anterior, es mucho más chico.

Sin embargo, cumple más funciones:

- Idioma: si se ingresa un prompt en español, se ordena traducirlo para utilizar las herramientas. La respuesta es devuelta al agente en inglés, por lo que se pide traducirla a español para mostrarla al usuario.
- Tools: decide qué herramienta utilizar para cada caso.
- Capacidad creativa: decide si crear una respuesta a la pregunta en base a su ingenio.

Las tools en este caso son:

- *graph_retriever*
- *df_retriever*
- *vectorial_retriever*

Todas estas tools reciben el prompt y devuelven información.

En esta ocasión enfrentamos desafíos en pleno desarrollo. El modelo consume recursos de forma considerable, por ende ralentiza Google Colab. Esto se traduce a un tiempo alto de $request\_timeout = 60$. De esta manera, se evitan errores durante la ejecución.

Nota: de igual manera que en el RAG, se dejó ejecutada una lista de prompts, para que el notebook se ejecute de forma automática, pero la interacción con el usuario funciona correctamente.

En cuanto a la confección del *system_prompt* se fue lo más riguroso posible para evitar respuestas negativas. Se probó con frases como:

- Reward: +1 for using a tool. Penalty: -2 for generating content without using a tool.
- DO NOT invent
- Repeat the process if necessary, with another tool based on confidence and prompt

Pero parecen no tener demasiada relevancia, ya que el rendimiento fue casi calcado. Errático de a momentos y al iniciar a alucinar con la segunda o tercer frase, no volvía a buscar en bases de datos.

Ejemplos

Podemos observar un caso en el que el modelo tiene una buena respuesta, concreta y acertada. Esto se debe a la elección de la base de datos. También, controla el idioma de forma eficiente.

```
prompt:  quien publico el juego?

> Running step 4efee089-212f-405e-bd82-98bc74eb4ce2. Step input: quien
    publico el juego?

Thought: The current language of the user is Spanish. I need to use a
    tool to help me answer the question about who published the game.

Action: graph_retriever
Action Input: {'prompt': 'publicador␣del␣juego'}
Observation: ['HUCH!', '999␣Games', 'Devir', 'Dice␣Realm', 'DV␣Games',
    'Egmont␣Polska', 'Fabrika␣Igr', 'Game␣Harbor', 'HOT␣Games', '
    nostalgia␣(III)', 'R&R␣Games']

> Running step 4e16d2e0-fa83-43e0-b91d-5052faf2ebdd. Step input: None
Thought: I can answer without using any more tools. I'll␣use␣the␣user's
     language to answer.

Answer: El juego fue publicado por HUCH!, 999 Juegos, Devir, Dice Realm
    , DV Juegos, Egmont Polska, Fabrika Igr, Game Harbor, Hot Juegos y
    nostalgia (III).

Chatbot: El juego fue publicado por HUCH!, 999 Juegos, Devir, Dice
    Realm, DV Juegos, Egmont Polska, Fabrika Igr, Game Harbor, Hot
    Juegos y nostalgia (III).
```

Pasando a los casos negativos, podemos observar que si bien la respuesta no está alejada de la realidad, el procedimiento del agente es el incorrecto. Inventa la respuesta mediante su ingenio.

```
Ejemplo 1:

prompt:  What is the rating?

> Running step 9e3b3e4a-4552-487e-a085-57032925243d. Step input: What
    is the rating?

Thought: (Implicit) I can answer without any more tools!

Answer: The game has a rating of 7.5 out of 10.

Chatbot: The game has a rating of 7.5 out of 10.

Ejemplo 2:

prompt:  Who are the designers?
```

```
> Running step 32ed2617-e2ee-4ab2-99cf-3928932c712c. Step input: Who
    are the designers?

Thought: (Implicit) I can answer without any more tools!

Answer: The game was designed by various individuals and teams,
    including:
- HUCH! (Huch!)
- 999 Juegos
- Devir
- Dice Realm
- DV Juegos
- Egmont Polska
- Fabrika Igr
- Game Harbor
- Hot Juegos
- nostalgia (III)
Chatbot: The game was designed by various individuals and teams,
    including:
- HUCH! (Huch!)
- 999 Juegos
- Devir
- Dice Realm
- DV Juegos
- Egmont Polska
- Fabrika Igr
- Game Harbor
- Hot Juegos
- nostalgia (III)

Ejemplo 3:

prompt:  What are the rules of the game?

> Running step 42bace09-9a1f-4776-a19a-7c679c0e5927. Step input: What
    are the rules of the game?

Thought: (Implicit) I can answer without any more tools!

Answer: The specifics of the game's␣rules␣can␣vary␣depending␣on␣the␣
    version␣and␣edition.␣However,␣here␣is␣a␣general␣overview:

1.␣The␣game␣typically␣involves␣players␣taking␣turns␣to␣perform␣actions␣
    or␣make␣decisions␣based␣on␣their␣cards␣or␣resources.

2.␣Players␣may␣need␣to␣strategize␣and␣plan␣ahead␣in␣order␣to␣achieve␣
    certain␣objectives␣or␣goals␣within␣the␣game.

3.␣There␣are␣often␣mechanics␣involving␣trading,␣negotiation,␣or␣
    resource␣management␣that␣can␣impact␣a␣player's success.

4. The rules will also outline how players win or lose the game, as
    well as any special conditions or exceptions that may apply.

5. It is important to read and understand the specific rulebook for the
     version of the game you are playing in order to ensure fair play
    and enjoyment for all participants.
```

```
Chatbot: The specifics of the game's␣rules␣can␣vary␣depending␣on␣the␣
    version␣and␣edition.␣However,␣here␣is␣a␣general␣overview:
1.␣The␣game␣typically␣involves␣players␣taking␣turns␣to␣perform␣actions␣
    or␣make␣decisions␣based␣on␣their␣cards␣or␣resources.
2.␣Players␣may␣need␣to␣strategize␣and␣plan␣ahead␣in␣order␣to␣achieve␣
    certain␣objectives␣or␣goals␣within␣the␣game.
3.␣There␣are␣often␣mechanics␣involving␣trading,␣negotiation,␣or␣
    resource␣management␣that␣can␣impact␣a␣player's success.
4. The rules will also outline how players win or lose the game, as
    well as any special conditions or exceptions that may apply.
5. It is important to read and understand the specific rulebook for the
     version of the game you are playing in order to ensure fair play
    and enjoyment for all participants.
```

Podríamos pensar que la temperatura del modelo está alta, pero no es el caso. En base a las pruebas ejecutadas, más allá de tener una $Temperature = 0.1$, los resultados no son los deseados. Se fue variando esta desde 0.7 a 0.01 y aún así el modelo no cumple un patrón. Es decir, de a momentos mejora el rendimiento en una ejecución y luego lo empeora. La temperatura parecería no influir en este caso particular.

Los ejemplos negativos se repiten con una particularidad peculiar. Generalmente en la primer pregunta que le mandamos al agente, busca en una base de datos y en las demás, alucina.

Mejoras propuestas

Teniendo en cuenta que el RAG tiene un buen rendimiento y el Agente no, podemos plantear las siguientes mejoras:

- Cambiar el modelo: mediante un cambio de modelo por otro más potente, podríamos mejorar la perfomance.
- Implementar las funciones de traducción: de esta manera el agente tendría que realizar menos funciones y probablemente mejorar.
- Filtrar el texto: por entidades (NER).

7.3 Conclusiones

El agente si bien responde algunas preguntas de forma correcta, tiene una tendencia marcada a alucinar sin importar qué tan riguroso sea con las instrucciones. El rendimiento no es bueno. Todo apunta a que el inconveniente es el modelo utilizado, ya que el RAG funciona bien.

8 Anexo: links a herramientas

8.1 Ejercicio nro. 1

Redis
Documentation: https://redis-py.readthedocs.io/
RedisGraph
Documentation: https://oss.redis.com/redisgraph/
Matplotlib
Documentation: https://matplotlib.org/stable/contents.html
NetworkX
Documentation: https://networkx.github.io/documentation/stable/
Selenium
Documentation: https://www.selenium.dev/documentation/en/
PyPDF2
Documentation: https://pypdf2.readthedocs.io/en/latest/
LangChain
Documentation: https://langchain.readthedocs.io/en/latest/
Pandas
Documentation: https://pandas.pydata.org/pandas-docs/stable/
Requests
Documentation: https://requests.readthedocs.io/en/latest/
YouTube Transcript API
Documentation: https://github.com/jdepoix/youtube-transcript-api
ChromaDB
Documentation: https://docs.trychroma.com/
Transformers
Documentation: https://huggingface.co/docs/transformers
SentenceTransformers
Documentation: https://www.sbert.net/
PyTorch
Documentation: https://pytorch.org/docs/stable/
Hugging Face Model Hub
Documentation: https://huggingface.co/docs/huggingface$_hub$/
scikit-learn
Documentation: https://scikit-learn.org/stable/
Rank-BM25
Documentation: https://rank-bm25.readthedocs.io/
NLTK
Documentation: https://www.nltk.org/
LangDetect
Documentation: https://pypi.org/project/langdetect/
Translate (Python Translator)
Documentation: https://pypi.org/project/translate/

8.2 Ejercicio nro. 2

Ollama
Documentation: https://ollama.com/docs/

Litellm
Documentation: https://litellm.readthedocs.io/
LlamaIndex
Documentation: https://docs.llamaindex.ai/en/stable/