



Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
T.U.I.A

Trabajo Práctico Final

Procesamiento del Lenguaje Natural

Integrante:
Lucía Masciangelo

Profesores:
Juan Pablo Manson
Alan Geary
Constantino Ferrucci
Dolores Sollberger

2025

Resumen	2
Introducción	3
Ejercicio 1 – RAG – Sistema de Recuperación con Generación Aumentada	4
Objetivo	4
Metodología	4
Desarrollo	6
Recuperación documental	6
Recuperación en datos tabulares	8
Recuperación en base de grafos	9
Modelo de clasificación	10
Pipeline de Recuperación del Sistema RAG	12
Resultados	13
Ejercicio 2 – Evolución del RAG: Agente Autónomo ReAct	17
Objetivo	17
Metodología	17
Desarrollo	18
Encapsulamiento de herramientas	18
Diseño del prompt de sistema	18
Implementación del agente ReAct	18
Manejo de múltiples herramientas	19
Memoria conversacional	19
Resultados	19
Conclusiones	23
Bibliografía	26

Sistema RAG y Agente autónomo

Resumen

Este trabajo desarrolla un sistema de conversación inteligente enfocado en el juego de mesa *Cascadia*, utilizando una arquitectura de tipo RAG (Retrieval-Augmented Generation) y evolucionando hacia un agente autónomo basado en el paradigma ReAct. Se integran tres tipos de fuentes de datos —textos informativos, estadísticas tabulares y relaciones entre entidades—, cada una accedida mediante herramientas específicas y consultadas dinámicamente para su posterior respuesta en lenguaje natural.

El sistema emplea modelos de lenguaje para interpretar preguntas del usuario, clasificarlas según el tipo de información requerida y recuperar la respuesta más pertinente desde múltiples fuentes. Para cada fuente, se definieron estrategias particulares: búsqueda híbrida y reranking para texto, generación de filtros automáticos para tablas, y conversión de preguntas en queries Cypher para grafos.

Además, se construyó un agente autónomo con LangChain que selecciona la herramienta más adecuada para cada consulta y es capaz de razonar sobre pasos intermedios.

Ambos modelos (RAG y Agentes) tienen incorporado memoria conversacional, y detección de idioma para responder en el mismo idioma en el que el usuario hace la consulta.

Introducción

En el marco del trabajo final de la asignatura *NLP*, se propone el desarrollo de un sistema de conversación especializado en el juego de mesa *Cascadia*, empleando técnicas modernas de generación aumentada por recuperación (RAG) y agentes autónomos con herramientas especializadas.

Este trabajo surge como continuación del TP1, donde ya se había generado y procesado una base documental diversa compuesta por textos, datos tabulados y relaciones entre entidades. En esta nueva instancia, se busca no solo recuperar esa información, sino también integrarla en una conversación fluida, contextualizada y multifuente, capaz de responder consultas específicas del usuario.

Los objetivos específicos fueron:

- Integrar diversas fuentes heterogéneas en un sistema único.
- Implementar clasificadores de intención y pipelines de recuperación con reranking.
- Desarrollar un agente inteligente con LangChain que decida qué herramientas utilizar.
- Evaluar el rendimiento mediante pruebas prácticas.

El presente informe describe detalladamente las decisiones técnicas tomadas, las herramientas utilizadas, y el proceso de desarrollo e integración del sistema, concluyendo con un análisis de resultados y sugerencias de mejora.

Ejercicio 1 – RAG – Sistema de Recuperación con Generación Aumentada

Objetivo:

Desarrollar un sistema de recuperación de información basado en la arquitectura RAG (Retrieval-Augmented Generation) que un usuario pueda realizar consultas en lenguaje natural sobre el juego *Cascadia*, y recibir respuestas precisas sin necesidad de conocer la estructura de los datos ni cómo acceder a ellos. Para lograrlo, se integran múltiples fuentes de datos (textos, tablas y grafos) y mediante técnicas de clasificación de intención, recuperación semántica, y generación de respuestas, se le otorga al usuario una respuesta en lenguaje natural.

Metodología:

El sistema RAG se construyó utilizando las tres fuentes de datos procesadas previamente en el TP1P1 incluyendo datos de:

- **Información textual** proveniente del manual del juego y foros.
- **Datos tabulados** con estadísticas de partidas y puntuaciones
- **Conjunto de relaciones** entre elementos del juego, y personas como diseñadores, ilustradores, y desarrolladores, expresadas en forma de triplas relacionando a Cascadia con algún otro juego o Persona.

El proyecto fue desarrollado en un entorno de Google Colab, utilizando Python como lenguaje principal.

Para construir la base de datos vectorial, se fragmentaron los documentos de texto utilizando el método **RecursiveCharacterTextSplitter**, el cual permite definir tanto la longitud máxima de cada fragmento como la superposición entre ellos. Esta técnica resulta especialmente útil para preservar la coherencia semántica dentro de cada chunk, algo crucial en tareas de recuperación basada en significado.

Los textos fueron vectorizados empleando el modelo **paraphrase-multilingual-mpnet-base-v2**, elegido por su excelente desempeño en generación de embeddings multilingües, lo que asegura una mayor robustez frente a consultas formuladas de distintas maneras y en

diferentes idiomas.

Los vectores generados se almacenaron en **ChromaDB**, una base de datos vectorial que se integra fácilmente con modelos de lenguaje y permite realizar consultas semánticas de forma eficiente.

Posteriormente, se implementó una estrategia de búsqueda híbrida que combina resultados obtenidos mediante **BM25** (búsqueda por coincidencia de términos) con aquellos basados en similitud semántica. Finalmente, se aplicó un proceso de **re-ranking** sobre los fragmentos recuperados, priorizando aquellos con mayor similitud de embeddings respecto a la consulta original, con el objetivo de mejorar la precisión de las respuestas.

En cuanto a los datos tabulares, se cargaron cada uno de los 4 disponibles en un DataFrame diferente y se elaboró un resumen automático de los campos disponibles (valores máximos, mínimos, únicos, etc.) sobre cada uno. Estos resúmenes se proporcionaron como contexto a un modelo de lenguaje natural para que pudiera generar filtros adecuados en Pandas según las consultas del usuario. Este enfoque evitó sobrecargar el modelo con todo el dataset y permitió respuestas más focalizadas y eficientes.

Las relaciones extraídas se representaron como triplas del tipo (sujeto1, relación, sujeto2) y fueron almacenadas en una base de datos de grafos implementada con **Neo4j**. Para facilitar la interacción con el grafo, se desarrolló una función que utiliza prompting con un modelo de lenguaje (LLM) para traducir preguntas formuladas en lenguaje natural a consultas válidas en **Cypher**, el lenguaje de consulta propio de Neo4j. Luego, los resultados obtenidos se procesan mediante el mismo modelo de lenguaje para convertirlos en respuestas comprensibles y legibles para el usuario final. Este enfoque permitió realizar consultas complejas sobre las relaciones del juego sin requerir conocimientos técnicos específicos por parte del usuario.

Por último, se implementó un **clasificador de intención** cuya función es identificar la fuente de información adecuada para cada consulta, distinguiendo entre texto documental, datos tabulares o grafo de relaciones.

Para ello, se evaluaron dos enfoques:

- por un lado, un modelo tradicional basado en la combinación de un vectorizador **FastText** y un clasificador **Random Forest**, previamente desarrollado en el TP1P2;
- y por otro, un enfoque basado en **prompting few-shot** con un modelo de lenguaje (LLM).

Este último se destacó por su mayor capacidad para interpretar consultas

ambiguas, mal formuladas o más conversacionales, motivo por el cual fue seleccionado como el clasificador principal del sistema.

A lo largo de todo el desarrollo del trabajo práctico se utilizó como modelo de lenguaje al modelo "**gemini-2.5-flash-preview-05-20**" de Gemini, ya que es un modelo gratuito que permite generar respuestas de calidad, con bajo tiempo de respuesta y compatibilidad nativa con herramientas de LangChain. Además, al estar optimizado para tareas conversacionales e instructivas, resultó adecuado tanto para el prompting de clasificación y recuperación, como para el razonamiento paso a paso dentro del agente ReAct. A partir de ahora cada vez que mencionemos un modelo LLM, vamos a hacer referencia a éste.

Desarrollo:

Para implementar el sistema RAG, se integraron tres módulos independientes de recuperación de información: uno basado en documentos textuales, otro sobre datos tabulados, y un tercero orientado a consultas en una base de grafos. La selección de qué módulo activar frente a cada consulta se resuelve a través de un clasificador de intención, lo que permite redirigir dinámicamente cada pregunta hacia la fuente más adecuada.

A continuación, se detalla el funcionamiento de cada componente:

- **Recuperación documental:**

Este componente responde a consultas relacionadas con aspectos generales del juego, como reglas, mecánicas, estrategias, o incluso experiencias compartidas por otros jugadores.

El proceso comenzó con una etapa de **limpieza de los textos** contenidos en la fuente de información. Dado que muchos de estos archivos provenían de procesos de web scraping, fue necesario eliminar fragmentos irrelevantes o ruido estructural (como menús, encabezados o scripts). Luego, se realizó una revisión individual más profunda de cada archivo para conservar únicamente los segmentos con información significativa y descartando el contenido irrelevante.

Una vez depurados los textos, se procedió a **fragmentarlos** utilizando la técnica **RecursiveCharacterTextSplitter**, que permite mantener la

coherencia semántica de cada unidad textual. Cada fragmento fue almacenado en un diccionario, donde la clave corresponde al nombre del archivo original y el valor al contenido del fragmento. Esta estructura resultó útil para poder identificar posteriormente el origen de cada segmento recuperado.

La **vectorización** de los fragmentos se llevó a cabo utilizando el modelo **paraphrase-multilingual-mpnet-base-v2**, elegido por su solidez en tareas de similitud semántica y su capacidad multilingüe. Los vectores generados fueron almacenados en **ChromaDB**, una base de datos vectorial eficiente y de integración sencilla, que permite además guardar los metadatos correspondientes a cada segmento, como por ejemplo el nombre del archivo original de donde proviene.

Para responder una consulta, el sistema implementa una **búsqueda híbrida**, combinando dos mecanismos: por un lado, similitud semántica entre embeddings; por el otro, coincidencia de términos mediante BM25. Dado que ambas puntuaciones operan en escalas distintas, se aplicó una **normalización sobre los resultados de BM25** antes de fusionarlos. Finalmente, se realiza un **re-ranking** que prioriza los fragmentos con mayor similitud global, generando así respuestas más precisas y relevantes para el usuario.

Para determinar el modelo de re-ranking más adecuado, se evaluaron cuatro variantes de **CrossEncoder** disponibles en Hugging Face:

- **cross-encoder/ms-marco-MiniLM-L-6-v2**: preciso y rápido (0.48s) (score 4.46).
- **cross-encoder/ms-marco-MiniLM-L-12-v2**: mayor precisión general pero mayor tiempo (0.95s) (score 4.58).
- **mixedbread-ai/mxbai-rerank-xsmall-v1**: menor desempeño, y tiempos más altos (2.51s) (score 0.67).
- **cross-encoder/ms-marco-TinyBERT-L-6-v2**: no disponible al momento de la evaluación.

Se priorizó el balance entre precisión y velocidad, así como la calidad de los fragmentos re-rankeados (más completos, detallados y con mejor cobertura semántica). Como resultado, se seleccionó el modelo **MiniLM-L-12-v2**, que demostró ser el más consistente en recuperar información útil en primeros puestos, manteniendo tiempos razonables para un entorno interactivo. Esta decisión se incorporó al pipeline final del sistema RAG.

Ejemplo: "Como se juega Cascadia?"

Cascadia es un juego de drafting (selección) de losetas y fichas para 1 a 4 jugadores. La dinámica principal consiste en que los jugadores irán seleccionando losetas y fichas, colocándolas en su tablero personal para crear una extensión geográfica.

El juego implica un puzzle de doble capa que progresa a medida que se añaden más hábitats y fauna (animales) al ecosistema individual de cada jugador. Para ganar puntos, los jugadores deben crear las disposiciones espaciales más armoniosas de la fauna y también los corredores de hábitat contiguos más grandes.

Y si hacemos la misma pregunta pero en inglés: "How do you play Cascadia?"

Cascadia is a tile and token drafting game played by 1 to 4 players. To play, players will draft tiles and tokens and then place them into their own individual ecosystem, also referred to as their tableau. The goal is to create a geographic expanse over time, which involves a dual-layer puzzle as you add more habitats and wildlife. Players earn end-game points by creating the most harmonious spatial arrangements of wildlife and by forming the largest contiguous habitat corridors.

Vemos, como ante la misma pregunta, el modelo genera la misma respuesta (o similar) en ambos idiomas, lo que indica que el modelo seleccionado es multilingüe y tiene una performance muy similar en ambos idiomas.

- **Recuperación en datos tabulares:**

Este módulo fue desarrollado para atender consultas relacionadas con información estructurada del juego, como estadísticas, conteos, comparaciones numéricas o atributos categóricos.

La fuente tabular está compuesta por cuatro archivos **.csv**, que fueron cargados individualmente en distintos **DataFrames** de Pandas. Como etapa inicial, se generó para cada archivo un resumen automático que extrae características clave: valores únicos en columnas categóricas, máximos y mínimos en columnas numéricas, y una breve descripción general de cada variable. Este resumen cumple una doble función: sirve como referencia para el modelo de lenguaje y permite limitar el contexto a lo estrictamente necesario.

Una vez cargado el resumen, se lo proporciona como parte del prompt a un modelo de lenguaje, que interpreta la intención de la consulta del usuario y genera automáticamente un filtro adecuado en Pandas. Este filtro se aplica sobre el DataFrame correspondiente, devolviendo una tabla ya reducida y enfocada en los datos relevantes para responder la pregunta.

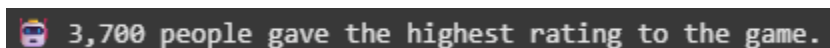
En una segunda etapa, esta tabla filtrada, junto con la pregunta original del usuario, se pasan nuevamente como entrada a otro modelo de lenguaje. Este último se encarga de **transformar los datos en una respuesta en lenguaje natural**, clara y comprensible, sin necesidad de que el usuario conozca estructuras de datos o sintaxis de programación.

En español: "¿Cuántas personas le dieron la puntuación más alta al juego?"



3700 personas le dieron la puntuación más alta al juego.

En inglés: "How many people gave the highest rating to the game?"



3,700 people gave the highest rating to the game.

En este caso, el modelo también sabe interpretar la pregunta del usuario en ambos idiomas (español, e inglés) generando la misma respuesta, lo que significa que puede entender ambos idiomas fácilmente

- **Recuperación en base de grafos:**

Este componente permite responder preguntas que involucran relaciones entre entidades del juego, como quiénes fueron sus diseñadores, ilustradores o desarrolladores, así como las conexiones entre distintas personas y otros juegos relacionados.

Las relaciones extraídas previamente se estructuraron en forma de triplas del tipo (sujeto1, relación, sujeto2) y se cargaron en **Neo4j**, una base de datos especializada en grafos que permite representar este tipo de información de forma natural y eficiente.

Para facilitar el acceso a esta base, se desarrolló una función que traduce preguntas formuladas en lenguaje natural a consultas **Cypher**, el lenguaje utilizado por Neo4j para operar sobre grafos. Esta traducción se realiza mediante prompting con un modelo de lenguaje, que genera dinámicamente la query correspondiente en función de la intención del usuario.

Una vez ejecutada la consulta sobre la base de grafos, los resultados obtenidos se reformulan nuevamente en lenguaje natural, generando una respuesta clara y comprensible. Este enfoque permite que el usuario pueda explorar relaciones complejas del juego sin necesidad de tener conocimientos técnicos ni escribir código especializado.

Español: "¿Cuáles son todos los juegos diseñados por Randy Flynn?"

Los juegos diseñados por Randy Flynn son: Cascadia: Rolling Rivers, Cascadia: Rolling Hills, Tabriz, Cascadia Junior y Cascadia.

Inglés: "What are all the games designed by Randy Flynn?"

Randy Flynn designed the following games: Cascadia: Rolling Rivers, Cascadia: Rolling Hills, Tabriz, Cascadia Junior, and Cascadia.

Una vez más, haciendo la prueba en los idiomas español e inglés, podemos ver como el modelo sabe interpretar y realiza la misma consulta y respuesta ante la misma pregunta pero en diferentes idiomas.

- **Modelo de clasificación:**

La decisión sobre qué módulo de recuperación utilizar frente a cada consulta se resuelve a través de un **clasificador de intención**, cuya función es identificar si la pregunta del usuario debe dirigirse a los documentos textuales, a los datos tabulares o a la base de grafos.

Para este componente se trabajaron dos enfoques:

- **Modelo tradicional:**

Este modelo fue desarrollado durante el TP1P2. Se probaron distintas combinaciones de vectorizadores (TF-IDF, CountVectorizer y FastText) junto con clasificadores clásicos (Naive Bayes, Logistic Regression y Random Forest). La mejor performance se obtuvo con la combinación de **FastText + Random Forest**.

Para entrenarlo, se generaron 100 preguntas para cada una de las tres fuentes de datos (información textual, estadísticas tabulares y relaciones), todas debidamente etiquetadas. El modelo alcanzó una **precisión del 94% (accuracy 0.94)** en validación interna. Además, se lo evaluó sobre un conjunto externo de seis preguntas nuevas no incluidas en el entrenamiento, que dieron un rendimiento bastante bajo.

```
How do you score points in the game Cascadia? => Información
Who designed the game Cascadia? => Información
What is the average final score per player? => Estadísticas
How is a keystone tile used? => Información
What other games did Randy Flynn design? => Información
How many people gave the game the highest rating? => Información
```

```
¿Cómo se ganan puntos en el juego Cascadia? => Estadísticas
¿Quién diseñó el juego Cascadia? => Relaciones
¿Cuál es el puntaje final promedio por jugador? => Estadísticas
¿Cómo se utiliza una loseta clave (keystone tile)? => Información
¿Qué otros juegos diseñó Randy Flynn? => Relaciones
¿Cuántas personas le dieron la puntuación más alta al juego? => Estadísticas
```

El entrenamiento de éste modelo se puede ver en profundidad en el siguiente enlace: [🔗 Entrenamiento.ipynb](https://colab.research.google.com/github/GoogleCloudPlatform/vertex-ai-samples/blob/main/notebooks/vertexai/How_to_use_Gemini_for_question_classification.ipynb)

- Modelo basado en LLM:

Como alternativa, se implementó un clasificador mediante **prompting** con un modelo de lenguaje (Gemini). Se adoptó un enfoque *few-shot*, presentando al modelo tres ejemplos por cada tipo de fuente (documental, tabular y grafo), y luego se le formularon las mismas seis preguntas de prueba utilizadas anteriormente. En este caso, el modelo clasificó correctamente **las seis consultas**, mostrando no solo mayor precisión, sino también una mejor comprensión del contexto y de posibles ambigüedades presentes en la redacción de las preguntas.

```
? ¿Cómo se ganan puntos en el juego Cascadia? ==> Información (0)
? ¿Quién diseñó el juego Cascadia? ==> Relaciones (1)
? ¿Cuál es el puntaje final promedio por jugador? ==> Estadísticas (2)
? ¿Cómo se utiliza una loseta clave (keystone tile)? ==> Información (0)
? ¿Qué otros juegos ilustró Randy Flynn? ==> Relaciones (1)
? ¿Cuántas personas le dieron la puntuación más alta al juego? ==> Estadísticas (2)
```

```
? How do you score points in the game Cascadia? ==> Información (0)
? Who designed the game Cascadia? ==> Relaciones (1)
? What is the average final score per player? ==> Estadísticas (2)
? How is a keystone tile used? ==> Información (0)
? What other games did Randy Flynn illustrate? ==> Relaciones (1)
? How many people gave the game the highest rating? ==> Estadísticas (2)
```

Dado este rendimiento ampliamente superior y su capacidad para generalizar sin necesidad de entrenamiento adicional, incluso en preguntas realizadas en

diferentes idiomas, se optó por utilizar el enfoque basado en LLM como clasificador principal del sistema.

- **Pipeline de Recuperación del Sistema RAG:**

Una vez definidos e implementados los componentes individuales (recuperación documental, tabular y relacional, junto con el clasificador de intención), se diseñó un **pipeline de recuperación** que permite integrarlos de forma dinámica y modular. Este pipeline constituye el núcleo funcional del sistema RAG, permitiendo responder consultas en lenguaje natural de manera eficiente y contextualizada.

El flujo comienza con la **entrada del usuario**, que formula una pregunta sobre el juego *Cascadia*. Esa consulta es procesada primero por el **clasificador de intención**, el cual determina a qué fuente de datos corresponde: documentos textuales, estadísticas tabulares o grafo de relaciones. En base al resultado, se redirige la consulta al módulo adecuado.

Cada módulo, a su vez, implementa su propia lógica de recuperación:

- En el caso de las **consultas textuales**, se realiza una búsqueda híbrida sobre la base vectorial (ChromaDB), seguida por un proceso de *re-ranking* para priorizar los fragmentos más relevantes. El contenido recuperado es finalmente enviado a un modelo de lenguaje que genera una respuesta legible para el usuario.
- Para las **consultas sobre datos tabulares**, se genera un resumen automático de la base, se utiliza un modelo de lenguaje para inferir los filtros correspondientes, y luego se crea un nuevo DataFrame filtrado con los datos solicitados. Esta tabla es enviada nuevamente a un LLM, que elabora una respuesta en lenguaje natural basada en esos datos.
- Cuando se trata de **consultas relacionales**, se emplea prompting para traducir la pregunta del usuario a una consulta Cypher válida, la cual se ejecuta sobre la base de grafos Neo4j. Los resultados se transforman luego en una respuesta textual comprensible.

Además, el pipeline cuenta con un **sistema de memoria conversacional**, lo que le permite mantener el contexto de preguntas anteriores. Gracias a esto, el usuario puede realizar consultas encadenadas o referirse a información

previamente mencionada sin necesidad de repetirla, mejorando significativamente la fluidez y naturalidad de la interacción.

Por otro lado, el sistema está diseñado para trabajar exclusivamente sobre el dominio del juego *Cascadia*. En caso de recibir preguntas ajenas a ese contexto —por ejemplo, “¿Quién es Messi?”—, el modelo detecta la falta de información relevante y responde indicando que no puede ofrecer una respuesta útil. Esto asegura que el chatbot mantenga el foco temático, evitando respuestas erróneas o fuera de lugar.

Este enfoque permite mantener cada componente aislado pero interrelacionado, favoreciendo tanto la reutilización como la escalabilidad.

Resultados:

A continuación se muestran algunos ejemplos del RAG en funcionamiento:

** Nota sobre el idioma de las pruebas:*

Las pruebas realizadas tanto en el sistema RAG como en el agente ReAct se llevaron a cabo mayoritariamente en español, por una cuestión de comodidad a la hora de interpretar las respuestas. Sin embargo, como se muestra en los ejemplos presentados, el sistema es plenamente capaz de comprender y responder preguntas formuladas en inglés, generando respuestas equivalentes y coherentes en ambos idiomas. Por ello, se optó por interactuar con los sistemas en español durante la evaluación, sin que esto represente una limitación técnica, sino una decisión orientada a la claridad en la validación de resultados.

🗨️ Asistente activado. Escriba su pregunta o 'salir' para cerrar.

🧠 Escriba su pregunta: ¿Cuántas personas calificaron al juego con 10?

📌 Clasificación de la pregunta: 2 (Estadísticas)

🗣️ 3700 personas calificaron el juego con 10.

🧠 Escriba su pregunta: ¿Y con 3?

📌 Clasificación de la pregunta: 2 (Estadísticas)

🗣️ 108 personas calificaron el juego con 3.

🧠 Escriba su pregunta: salir

👋 ¡Hasta luego!

🗨️ Asistente activado. Escriba su pregunta o 'salir' para cerrar.

🧠 Escriba su pregunta: ¿Cuándo se gana el juego?

📌 Clasificación de la pregunta: 0 (Información)

🗣️ El juego se gana cuando el último jugador ha completado su turno número 20. En ese momento, la partida finaliza, se puntúan las cartas de animales y los hábitats continuos más grandes, y el jugador que ha acumulado la mayor cantidad de puntos es el ganador.

🧠 Escriba su pregunta: ¿Cómo se ganan los puntos?

📌 Clasificación de la pregunta: 0 (Información)

🗣️ En Cascadia, los puntos se ganan principalmente de dos maneras:

1. ****Al colocar las fichas de animales:**** Ganas puntos si las colocas en las losetas de territorio correctas, siguiendo las condiciones y especificaciones de puntuación que cada especie animal tiene (hay cuatro cartas de puntuación por cada una, ofreciendo muchas combinaciones).

2. ****Al construir los hábitats continuos más grandes:**** También obtienes puntos por las áreas más grandes que logres construir con las losetas de terreno.

🧠 Escriba su pregunta: salir

🧠 Escriba su pregunta: ¿Quién diseñó Cascadia?

 Clasificación de la pregunta: 1 (Relaciones)

  Respuesta:

Cascadia fue diseñado por Randy Flynn.

 Escriba su pregunta: ¿Y qué otros juegos diseñó esta persona?


 Clasificación de la pregunta: 1 (Relaciones)


  Respuesta:


Randy Flynn diseñó los siguientes juegos: Cascadia: Rolling Rivers, Cascadia: Rolling Hills, Tabriz, Cascadia Junior y Cascadia.


 Escriba su pregunta: salir


 ¡Hasta luego!


 Asistente activado. Escriba su pregunta o 'salir' para cerrar.


 Escriba su pregunta: ¿Quien es Messi?


 No se pudo clasificar: No se puede responder con los datos del juego, por favor reformule la pregunta o haga una distinta.


 No se pudo clasificar la pregunta. Reformulá por favor.


 Escriba su pregunta: salir


 ¡Hasta luego!


 Asistente activado. Escriba su pregunta o 'salir' para cerrar.


 Escriba su pregunta: ¿Cuál es el puntaje promedio del juego?


 Clasificación de la pregunta: 2 (Estadísticas)


 El puntaje promedio del juego es 7.918.

 Escriba su pregunta: ¿Cuántas personas reaccionaron al juego?

 Clasificación de la pregunta: 2 (Estadísticas)

 La cantidad de personas que reaccionaron al juego es 47066.

 Escriba su pregunta: salir

 ¡Hasta luego!

🗨 Asistente activado. Escriba su pregunta o 'salir' para cerrar.

💡 Escriba su pregunta: ¿Qué otros juegos ilustró la persona que ilustró Cascadia?

📌 Clasificación de la pregunta: 1 (Relaciones)

🤖 La persona que ilustró Cascadia también ilustró los siguientes juegos:

- * Wingspan
- * Viticulture Essential Edition
- * Calico
- * Viticulture
- * Arboretum

Gracias a esta **arquitectura modular y automatizada**, el sistema es capaz de responder con precisión a una amplia variedad de preguntas sobre el juego de mesa Cascadia, seleccionando de forma dinámica y autónoma la fuente de información más adecuada en cada caso. Esta organización no solo facilita el mantenimiento y escalabilidad del sistema, sino que también garantiza una experiencia de usuario fluida y coherente, sin necesidad de intervención manual por parte del desarrollador o del usuario.

Ejercicio 2 – Evolución del RAG: Agente Autónomo ReAct

Objetivo:

El objetivo de este ejercicio fue transformar el sistema RAG modular en un **agente conversacional autónomo** capaz de tomar decisiones sobre qué herramienta utilizar para responder preguntas complejas en lenguaje natural. Para ello, se implementó un agente basado en el paradigma **ReAct (Reasoning + Acting)**, utilizando la biblioteca **LangChain**.

El agente no solo puede recuperar información desde diferentes fuentes (documentos, tablas, grafos), sino que también es capaz de **razonar, elegir acciones, encadenarlas y generar una respuesta completa**, manteniendo el contexto conversacional y trabajando de forma dinámica.

Metodología:

Para construir el agente se reutilizaron los tres componentes ya desarrollados en el ejercicio anterior (**consultas textuales, consultas sobre datos tabulares, consultas relacionales**) y se encapsularon como herramientas independientes (**doc_search, table_search, y graph_search** respectivamente). A estas se sumaron dos herramientas adicionales:

- **wikipedia_search()**: para consultas generales que no estén cubiertas por las fuentes propias del sistema.
- **web_search()**: para búsqueda abierta en la web como último recurso, que no se encuentren ni en las propias fuentes, ni en wikipedia.

Cada herramienta fue registrada dentro del agente como una *Tool*, siguiendo la estructura que propone LangChain. Se diseñó un **prompt de sistema** que instruye al modelo sobre el uso correcto de cada herramienta, y lo guía para que actúe según el patrón ReAct: primero pensar (*Thought*), luego actuar (*Action*), y finalmente observar los resultados (*Observation*), iterando si es necesario, y volver a buscar a otra fuente de no encontrar una respuesta que le parezca adecuada al sistema.

Al igual que en el RAG, se empleó memoria conversacional para preservar el contexto del usuario a lo largo de múltiples turnos, y se garantizó que el agente priorice las fuentes propias antes de recurrir a fuentes externas, con el objetivo de mantener el foco en el juego *Cascadia*.

Desarrollo:

El desarrollo del agente se estructuró en varias etapas:

- **Encapsulamiento de herramientas:**

Cada función de búsqueda en las diferentes fuentes de datos fue transformada en una función independiente que acepta inputs estándar (consulta o filtros) y devuelve una respuesta textual. Estas funciones se registraron como tools dentro del agente de LangChain.

- **Diseño del prompt de sistema:**

Se creó un prompt específico que incluye instrucciones claras sobre:

- El propósito del agente.
- Qué tipo de consultas maneja cada herramienta.
- En qué orden deben priorizarse.
- Cómo debe razonar antes de ejecutar una acción.
- Cómo debe responder al usuario

- **Implementación del agente ReAct:**

Se utilizó un modelo de lenguaje (Gemini) como LLM central. El agente sigue un flujo estructurado de razonamiento y acción, donde:

- Interpreta la pregunta.
- Decide cuál o cuáles herramientas usar.
- Ejecuta las acciones necesarias.
- Integra las observaciones para generar una respuesta completa.
- Responde en el mismo idioma en el que el usuario realizó la pregunta.

- **Manejo de múltiples herramientas:**

El agente está preparado para encadenar acciones si una sola herramienta no es suficiente. Por ejemplo, puede usar `graph_search` para encontrar una entidad, y luego `doc_search` para obtener más información sobre ella.

- **Memoria conversacional:**

Se integró un mecanismo de memoria que permite que el agente recuerde preguntas previas, referencias a entidades mencionadas antes, y mantenga coherencia a lo largo de la interacción.


Resultados:

A continuación se muestran algunos resultados de las interacciones con el agente.

Al igual que con el RAG, Se interactuó en español por una facilidad de interpretación de las consultas.

```
 Bienvenido al asistente experto en Cascadia.
```

```
 Ingrese su pregunta o escriba 'salir' para finalizar.
```

```
 Pregunta del usuario: ¿Quien gana el juego Cascadia?
```

```
> Entering new AgentExecutor chain...
```

```
Action: doc_search
```

```
Action Input: ¿Quien gana el juego Cascadia?
```

```
Observation: Según el contexto proporcionado, no se especifica quién gana el juego Cascadia.
```

```
El texto describe el juego como un juego de colocación de losetas donde los jugadores construyen su terreno y lo pueblan con vida silvestre, creando un ecosistema diverso y armonioso. También menciona que cada especie animal tiene una preferencia espacial diferente y que los hábitats deben colocarse para reducir la fragmentación y crear corredores de vida silvestre, lo que sugiere un sistema de puntuación o cumplimiento de objetivos, pero no detalla explícitamente cómo se determina el ganador.
```

El contexto se enfoca en que Cascadia ganó el premio "Spiel des Jahres" en 2022 y otros premios, además de mencionar su creador y comparaciones con otros juegos.

Thought: Action: doc_search

Action Input: Cascadia reglas de puntuación

Observation: Según el contexto proporcionado, las reglas de puntuación en Cascadia se definen al inicio de cada partida mediante cartas de puntuación.

Funciona así:

- * Para cada especie animal, hay cuatro cartas de puntuación diferentes.

- * Al comienzo de la partida, se elige una de estas cartas (ya sea al azar o por decisión conjunta de los jugadores) para esa sesión de juego.

- * Esta carta de puntuación es la que determina cómo se deben colocar los marcadores de dicha especie en tu paisaje para poder ganar puntos.

Thought: Action: doc_search

Action Input: Cascadia cómo se determina el ganador

Observation: En Cascadia, el ganador se determina por la cantidad de puntos que cada jugador acumula.

Estos puntos se consiguen principalmente a través de **tarjetas de puntuación** específicas para cada especie de animal. Al inicio de cada partida, se seleccionan estas tarjetas (hay cuatro diferentes para cada animal), y **ellas definen cómo debes colocar los animales en tu paisaje para maximizar tu puntuación**. De esta manera, cada partida puede tener estrategias de puntuación diferentes según las tarjetas elegidas.

Thought: Final Answer: En Cascadia, el ganador es el jugador que acumula la mayor cantidad de puntos. Los puntos se obtienen principalmente a través de tarjetas de puntuación específicas para cada especie de animal, las cuales se seleccionan al inicio de cada partida y definen cómo se deben colocar los animales en el paisaje para maximizar la puntuación.

> Finished chain.

 Respuesta del agente:

En Cascadia, el ganador es el jugador que acumula la mayor cantidad de puntos. Los puntos se obtienen principalmente a través de tarjetas de puntuación específicas para cada especie de animal, las cuales se seleccionan al inicio de cada partida y definen cómo se deben colocar los animales en el paisaje para maximizar la puntuación.

 Pregunta del usuario: ¿Cuántas personas puntuaron el juego

> Entering new AgentExecutor chain...


Action: *table_search*

Action Input: *¿Cuántas personas puntuaron el juego?*

Observation: *47066 personas puntuaron el juego.*

Thought: *Final Answer: 47066 personas puntuaron el juego.*

> Finished chain.

 Respuesta del agente:

47066 personas puntuaron el juego.

 Pregunta del usuario: ¿Y cuántas lo puntuaron con 10?

> Entering new AgentExecutor chain...

Action: *table_search*

Action Input: *¿Cuántas veces se puntuó con 10?*

Observation: *⚠ Error: se esperaba un DataFrame o Series pero se recibió <class 'numpy.float64'>*

Thought: *Thought: The previous attempt to use `table_search` resulted in an error indicating that the tool returned a single numerical value (numpy.float64) when a DataFrame or Series was expected. This suggests an issue with how the tool's output is handled when it returns a direct*

count. However, ``table_search`` is the correct tool for obtaining numerical statistical data like counts. I need to re-attempt the query, phrasing it as directly as possible to ask for the count of scores of 10. If the error persists, it indicates a limitation of the tool's output format for single numerical results.

Action: `table_search`

Action Input: `cantidad de puntuaciones de 10`


Observation: `La cantidad de puntuaciones de 10 es 3700.`

Thought: `Final Answer: La cantidad de puntuaciones de 10 es 3700.`

> Finished chain.

 Respuesta del agente:

La cantidad de puntuaciones de 10 es 3700.

 Pregunta del usuario: ¿Qué otros juegos ilustró la persona que ilustró Cascadia?

> Entering new AgentExecutor chain...

Action: `graph_search`

Action Input: `¿Quién ilustró Cascadia?`

Observation: `Beth Sobel ilustró Cascadia.`

Thought: `Action: graph_search`

Action Input: `¿Qué otros juegos ilustró Beth Sobel?`

Observation: `Otros juegos que ilustró Beth Sobel son: Wingspan, Viticulture Essential Edition, Calico, Viticulture, Arboretum y Cascadia.`

Thought: `I now know the final answer`

Final Answer: `Beth Sobel ilustró Cascadia. Otros juegos que ilustró son Wingspan, Viticulture Essential Edition, Calico, Viticulture y Arboretum.`

```
> Finished chain.
```

 Respuesta del agente:

Beth Sobel ilustró *Cascadia*. Otros juegos que ilustró son *Wingspan*, *Viticulture Essential Edition*, *Calico*, *Viticulture* y *Arboretum*.

Conclusiones

El desarrollo del sistema RAG y su posterior evolución hacia un agente autónomo demostró ser una estrategia efectiva para construir un asistente conversacional experto enfocado en un dominio específico, en este caso el juego de mesa *Cascadia*.

Desde el punto de vista funcional, el **sistema RAG** logró resolver correctamente consultas de distinto tipo que se encontraban en las bases de datos: explicaciones reglamentarias, reglas de puntuación, promedios, autores del juego y relaciones entre personas y juegos. Gracias a la clasificación automática de intención y la integración modular de los tres motores de recuperación (documentos, tablas y grafos), el sistema fue capaz de seleccionar la fuente de datos adecuada en cada caso, sin intervención manual. Además, al incorporar una búsqueda híbrida con *re-ranking*, se mejoró la relevancia de los fragmentos textuales recuperados.

Las pruebas realizadas evidencian la capacidad del sistema RAG para responder con precisión preguntas específicas como “¿Cuándo se gana el juego?”, “¿Cómo se ganan los puntos?” o “¿Qué juegos diseñó Randy Flynn?”. Asimismo, se observaron aciertos significativos en la gestión del contexto y la memoria conversacional, como en el caso en que el usuario preguntó “¿Y qué otros juegos diseñó esta persona?” inmediatamente después de consultar por el autor de *Cascadia*, obteniendo una respuesta coherente sin necesidad de repetir la información.

Otro aspecto destacable es el comportamiento del sistema frente a consultas fuera de dominio. Ante preguntas como “¿Quién es Messi?”, el sistema RAG respondió correctamente indicando que la consulta no podía ser clasificada, sugiriendo reformular la pregunta. Esto evidencia una correcta **encapsulación del dominio**, ya que el sistema solo accede a información que forma parte

explícita de las bases de datos integradas y evita generar respuestas especulativas o irrelevantes.

En conjunto, esto demuestra que el sistema no solo es preciso dentro de su dominio, sino también robusto frente a entradas no pertinentes, reforzando su confiabilidad como asistente experto temático.

La evolución hacia el **Agente ReAct** potenció aún más la autonomía y flexibilidad del sistema. El agente demostró una capacidad superior para **razonar paso a paso**, elegir herramientas de manera dinámica y combinar múltiples fuentes en una misma conversación. Se observaron casos donde el agente recurrió secuencialmente a **doc_search**, **graph_search** y **table_search**, integrando sus resultados en una única respuesta coherente.

Por ejemplo, ante la consulta *“¿Quién ilustró Cascadia?”*, *“¿Y qué otros juegos ilustró esa persona?”*, el agente primero recuperó el nombre desde la base de grafos y luego generó correctamente una nueva consulta para obtener la lista de obras ilustradas por esa persona. Asimismo, supo manejar datos cuantitativos como *“¿Cuántas personas puntuaron el juego?”* y *“¿Cuántas le dieron un 10?”*, validando la funcionalidad del acceso tabular.

En cuanto a las limitaciones, se identificó un error en la primera ejecución de una consulta tabular que devolvió un valor **float** en lugar de un **DataFrame**. Sin embargo, el agente, no se conformó con esa respuesta, y fue capaz de **reintentar automáticamente la consulta** con una mejor formulación, mostrando un comportamiento resiliente.

A diferencia del sistema RAG, el agente ReAct cuenta con herramientas de búsqueda externa (**wikipedia_search** y **web_search**), lo que le permite salir del dominio cerrado del juego *Cascadia* cuando la situación lo requiere. Gracias a estas herramientas, el agente fue capaz de responder adecuadamente a consultas que no estaban contempladas en las bases de datos internas, como por ejemplo *“¿Quién es Messi?”*, accediendo a información actualizada desde la web y brindando una respuesta concreta y contextualizada.

Este comportamiento demuestra la **versatilidad del agente**, que puede operar como un asistente especializado pero, cuando es necesario, **extiende su alcance hacia fuentes externas**.

En conjunto, los resultados demuestran que tanto el sistema RAG como su versión evolucionada en forma de agente ReAct cumplen con los objetivos

propuestos: responden en lenguaje natural, consultan múltiples fuentes estructuradas, priorizan la precisión y mantienen el contexto.

En definitiva, si bien tanto el sistema RAG como el agente ReAct cumplen con los objetivos propuestos —mantener una conversación fluida en lenguaje natural, preservar el contexto entre turnos y responder adecuadamente sobre el juego *Cascadia*—, el agente presenta una **mayor capacidad de razonamiento**, ya que es capaz de replantear su estrategia cuando la primera respuesta no es satisfactoria y puede abordar **preguntas más complejas** gracias a su arquitectura iterativa.

No obstante, esta flexibilidad conlleva un riesgo: al contar con herramientas externas como Wikipedia o búsqueda web, el agente puede acceder a información fuera del dominio del juego incluso cuando el usuario no lo solicita explícitamente. Esto puede llevar a respuestas incorrectas o fuera de foco si las bases propias no contienen suficiente información. Sin embargo, esta misma característica también puede ser una **ventaja en contextos donde se busca reducir la dependencia de bases de datos exhaustivas o especializadas**, ya que permite extender el conocimiento del sistema sin necesidad de cargar manualmente todos los contenidos.

En síntesis, si el objetivo es construir un asistente altamente especializado, que responda **únicamente** sobre información previamente curada y controlada, el enfoque basado en RAG resulta más adecuado. En cambio, si se prioriza la versatilidad, la flexibilidad, el razonamiento dinámico y la cobertura amplia de preguntas, el agente ReAct ofrece una solución más flexible y poderosa.

Como mejora, podría integrarse un sistema de validación del dominio antes de usar Wikipedia o WebSearch, para evitar desvíos innecesarios

En ambos casos, cabe destacar un principio transversal a todo el proyecto: **la calidad del prompting es fundamental**. La capacidad de un modelo de lenguaje para comprender, clasificar e interpretar correctamente las consultas del usuario depende en gran medida de cómo se le formula la tarea. Sin un buen prompt, no hay arquitectura ni modelo que garantice comprensión ni respuestas útiles, el modelo no puede razonar correctamente ni generar respuestas útiles, sin importar cuán avanzado sea su entrenamiento.

Bibliografía

HuggingFace - Modelos CrossEncoder: <https://huggingface.co/cross-encoder>

LangChain - Documentation: <https://docs.langchain.com/>

ChromaDB - Open-source Vector Database: <https://www.trychroma.com/>

Neo4j Cypher Query Language: <https://neo4j.com/docs/cypher-refcard/current/>

Gemini Model by Google: <https://ai.google.dev/api?hl=es-419&lang=python>

SBERT - Sentence Transformers: Multilingual Models:
https://www.sbert.net/docs/sentence_transformer/pretrained_models.html#multilingual-models