

Trabajo Práctico Final - Procesamiento del Lenguaje Natural

Arce, Sofía; A-4517/9

Universidad Nacional de Rosario

Profesores: *Manson, Juan Pablo - Geary, Alan - Ferrucci, Constantino - Sollberger, Dolores*

29 de junio, 2025

Indice:

Resumen	3
Introducción	4
Metodología	4
Desarrollo	4
Entorno de Trabajo y Fuentes de Datos:	5
Ejercicio 1	6
Creación de la Base de Datos Vectorial	6
Creación de los acceso a los datos estadísticos	7
Creación de la Base de Datos de Grafos	9
Implementación de un Clasificador Avanzado	10
Definición de un Pipeline de Recuperación	11
Integración para la Generación y Conversación:	13
Ejercicio 2	14
Resultados y Pruebas	15
Ejercicio 1	15
Ejercicio 2	18
Conclusiones	19
Referencias	20

Resumen

En el siguiente informe, se encuentra explicada la implementación y resolución de los ejercicios 1 y 2 del trabajo práctico de la materia. La sección de metodología contiene como fui estructurando la resolución de cada punto, luego en el desarrollo se divide en dos secciones: Ejercicio 1 y Ejercicio 2; ambas contienen las justificaciones de las herramientas elegidas y las técnicas utilizadas. Finalmente, en la parte de resultados están las capturas de pantalla de los resultados del archivo .ipynb incluido en el repositorio de la entrega y la sección de conclusión contiene mis opiniones del desarrollo, los resultados y posibles mejoras.

Introducción

En este trabajo final se consolidaron los datos extraídos de la primera parte y el análisis de la segunda, con la finalidad de implementar un chatbot experto en el juego Sagrada. Se implementó una arquitectura de *Retrieval-Augmented Generation* en el ejercicio 1 y un Agente ReAct para el segundo apalancado por las funciones definidas en la resolución del 1. En ambos casos, los chatbots interpretan correctamente las preguntas y generan respuestas coherentes con la información encontrada o bien le piden al usuario que vuelva a escribir su prompt.

Metodología

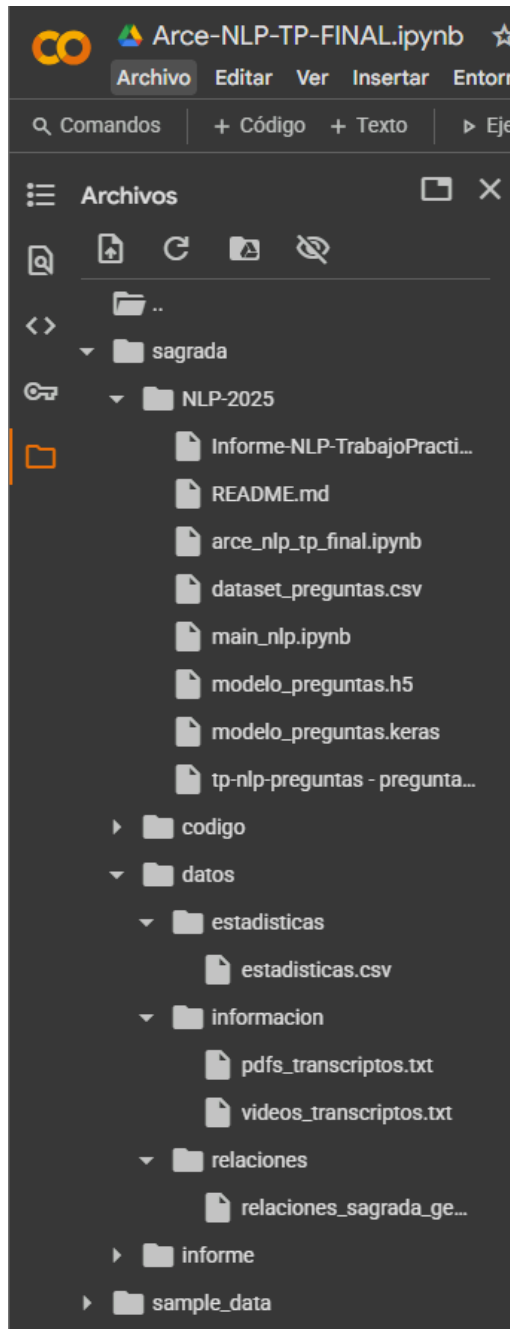
Para cada ítem planteado en la consigna, se estructuró el trabajo siguiendo principios de desarrollo ágil. Cada ítem desarrollado en la consigna, previo a su resolución, se subdividieron en tareas técnicas específicas a seguir para su implementación. Esta división, tuvo la finalidad de mantener un orden en el flujo de trabajo, facilitar el seguimiento y asegurarse de que cada ítem esté resuelto.

Desarrollo

Con la finalidad de cargar los datos dinámicamente al correr el archivo, se agregaron dos celdas para clonar dos repositorios: uno que corresponde a la primera entrega del trabajo que contiene la información y otro correspondiente a toda la materia con los archivos necesarios de la segunda entrega. En toda la resolución no se agregó información que no estuviese previamente en estos repositorios, es decir, no se incluyeron más datos de los extraídos previamente en las otras partes del trabajo práctico.

Entorno de Trabajo y Fuentes de Datos:

Todo el TP fue resuelto en el archivo Arce-NLP-TP-FINAL.ipynb, la estructura de los directorios es de la siguiente jerarquía:



Al comienzo del colab se clonan dos repositorios:

- **sagrada**: que contiene la resolución de la parte 1 del TP. El directorio utilizado es el de “datos”.
 - estadísticas, con el csv de las estadísticas extraídas en la primera etapa.
 - informacion, con los pdfs y transcripciones de los videos.
 - Relaciones, con las relaciones extraídas.
- **NLP-2025**: que corresponde a mi repositorio personal donde están los datos necesarios para la resolución de esta parte final. Para facilitar el desarrollo se clonó el repositorio completo, pero únicamente se utiliza el modelo entrenado en la etapa anterior (modelo_preguntas.h5) y el dataset de preguntas con la cual ese modelo fue entrenado (dataset_preguntas.csv).

Ejercicio 1

Creación de la Base de Datos Vectorial

En primera instancia, fue necesario hacer un preprocesamiento de los archivos para asegurarnos de que la información esté lo más limpia posible, para ello implementé dos funciones que se encargan del preprocesamiento de los archivos del PDF y otro para el de las transcripciones de los videos. Luego, los textos limpios se dividieron en chunks usando el modelo de Sentence Transformers **SpacyTextSplitter** la razón por la cual lo elegí fue porque quise preservar un poco el contexto de los texto y este método puede separar los chunks basándose en la estructura lingüística del texto, asegura de que cada chunk sea semánticamente relevante y preserve la integridad de la oración. Antes de crear los vectores, a cada chunk se le agregó metadata y se guardó como un objeto **Document** de **langchain** para luego agregarlos al índice. Finalmente, para la creación de los embeddings de cada chunks use el modelo de **all-mpnet-base-v2**, según la página de Hugging Face del modelo(1), el principal fin de éste es para generar embeddings de oraciones y párrafos cortos. El vector resultante puede ser usado para recuperación de la información (retrieval information), haciéndolo ideal para la resolución de este punto.

Una vez creados los chunks y definido el modelo de embeddings, seleccioné el motor de la base de datos vectorial: **Pinecone**, fue elegida porque, en mi opinión, tiene una documentación mucho más completa, con notebooks de ejemplos, casos de uso etc. a diferencia de otros motores como ChromaDB. Ahora con todo ya listo, se generaron IDs para almacenar cada chunk con su metadata, es decir, se agregaron distintas etiquetas para facilitar la búsqueda del origen de los datos (transcripción de video o información de pdfs). Para almacenarlos, se instanció la clase

PineconeVectorStore importada de la librería **langchain_pinecone** diseñada para interactuar con los índices de Pinecone y los el modelo de embeddings también importado de la librería de langchain **langchain_community.embeddings**, luego se agregaron los chunks con sus respectiva metadata a la BBDD con **add_documents (2)**

Para la búsqueda de los textos más parecidos, se usó **similarity_search(2)** que acepta texto libre como consulta, lo convierte automáticamente en embedding usando el modelo configurado, y realiza una consulta al índice de **Pinecone**. El resultado devuelto es una lista de objetos **Document** de **LangChain** que representan los textos más similares a la consulta

Creación de los acceso a los datos estadísticos

En esta parte, se presentó uno de los mayores desafíos porque en los datos tabulares la información se encontraba dentro de los registros junto con su valor:

df_estadísticas

	GAME STATS	GAME RANKS	PLAY STATS	COLLECTION STATS	PARTS EXCHANGE
0	Avg. Rating 7.472	Overall Rank 213 Historical Rank	All Time Plays 287,557	Own 73,808	Has Parts 19
1	No. of Ratings 44,391	Abstract Rank 10 Historical Rank	This Month 299	Prev. Owned 4,901	Want Parts 19
2	Std. Deviation 1.16	Family Rank 45 Historical Rank	NaN	For Trade 606 Find For-Trade Matches	NaN
3	Weight 1.92 / 5	NaN	NaN	Want In Trade 832 Find Want-in-Trade Matches	NaN
4	Comments 6,137	NaN	NaN	Wishlist 9,323	NaN
5	Fans 2,230	NaN	NaN	NaN	NaN
6	Page Views 1,851,300	NaN	NaN	NaN	NaN

Entonces fue necesario hacer una transposición del data frame para estructurar mejor las queries quedando así:

	Category	Metric	Value
0	GAME STATS	Avg. Rating	7.472
1	GAME STATS	No. of Ratings	44391.000
2	GAME STATS	Std. Deviation	1.160
3	GAME STATS	Weight	1.920
4	GAME STATS	Comments	6137.000
5	GAME STATS	Fans	2230.000
6	GAME STATS	Page Views	1851300.000
7	GAME RANKS	Overall Rank	213.000
8	GAME RANKS	Abstract Rank	10.000
9	GAME RANKS	Family Rank	45.000
10	PLAY STATS	All Time Plays	287557.000
11	PLAY STATS	This Month	299.000
12	COLLECTION STATS	Own	73808.000
13	COLLECTION STATS	Prev. Owned	4901.000
14	COLLECTION STATS	For Trade	606.000
15	COLLECTION STATS	Want In Trade	832.000
16	COLLECTION STATS	Wishlist	9323.000
17	PARTS EXCHANGE	Has Parts	19.000
18	PARTS EXCHANGE	Want Parts	19.000

Una vez transpuesta la estructura, se generó una lista con las posibles claves para darle contexto al chatbot. El llm elegido fue el modelo de gemini-2.5-flash por la buena cantidad de token diarios gratis y la abundante documentación con ejemplos.

En el contexto se le pasan las posibles combinaciones de las claves para que el llm genere las queries adecuadas para encontrar la respuesta (query_stats):

```

➡️ {'COLLECTION STATS': ['Own',
    'Prev. Owned',
    'For Trade',
    'Want In Trade',
    'Wishlist'],
    'GAME RANKS': ['Overall Rank', 'Abstract Rank', 'Family Rank'],
    'GAME STATS': ['Avg. Rating',
    'No. of Ratings',
    'Std. Deviation',
    'Weight',
    'Comments',
    'Fans',
    'Page Views'],
    'PARTS EXCHANGE': ['Has Parts', 'Want Parts'],
    'PLAY STATS': ['All Time Plays', 'This Month']}
```


Creación de la Base de Datos de Grafos

The diagram is a radial network with 'Sagrada' at the center. Nodes are arranged in concentric rings. Relationships are indicated by colored lines and labels: blue for 'Family', green for 'Publishers', pink for 'Designers', and brown for 'Mechanisms'. The nodes include various game titles and publishers, such as 'Mandoo Games', 'MINDOK', 'Daryl Andre...', 'Theme: Art', 'Devir', 'Galápagos', 'End Game Bon...', 'Cranio Creati...', 'Floodgate Games', 'Matagot', 'Engames', 'FoxGam...', 'Adrian Adam...', 'Pegasus Spiele', 'MEBO Games', 'Lex Games', 'Geekach Games', 'Lanlalen', 'Dice', 'One Moment Ga...', 'Dice Rolling', 'Turn Order: Pro...', 'MIPL', 'Solo / Solitaire Ga', 'Grid Cover...', 'Reflexsh...', 'Set Collec...', 'Open Drafting', 'Compon...', 'Contracts', 'Puzzle', 'Variable Set-up', 'Crowdfu...', 'Category: Tu...', 'Mechani...', 'Pattern Building', 'Digital Imple...', 'Misc: Watch It Play', 'Game: Sagra...', 'Organiz...', 'Cities: Barcelona', and 'Mandoo Games'.

Se pueden observar todos los nodos y sus aristas con los nombres de las relaciones entre ellos y el juego. En la información fuente no se tienen datos que relacionen a los nodos entre sí, es decir, sólo están relacionadas al juego/nodo “Sagrada” .

Para generar la interfaz del modelo que genere las queries de consulta en el data frame usé algo similar al punto anterior: en el contexto se le pasa al llm las posibles claves de las relaciones, ya que al ser generativo podría generar relaciones que no estén y de ese modo no obtener resultados del dataframe.

Implementación de un Clasificador Avanzado

Para el modelo clasificador avanzado, descargué el modelo entrenado y sus datos desde el repositorio de un compañero al cual se le asignó este juego(3). Al llm, lo evalué en dos instancias para el few-shot (en cada una de ellas, reinicié el modelo):

- Sin consigna, simplemente una pregunta de cada categoría y su clasificación:

```
¿Qué celebridades han jugado Sagrada?: relaciones\n\n
Ejemplo 2:\n\n
¿Qué porcentaje de jugadores recomiendan Sagrada? // estadísticas\n\n
Ejemplo 3:\n\n
¿Cómo se eligen los dados al inicio del juego?//informacion"
```

- Luego con consigna + preguntas + clasificación:

```
prompt="\n\n
SOLAMENTE Clasifica las siguientes preguntas en las posibles categorías donde estará la información: relaciones, estadísticas o información:\n\n
Ejemplo 1:\n\n
¿Qué celebridades han jugado Sagrada?: relaciones\n\n
Ejemplo 2:\n\n
¿Qué porcentaje de jugadores recomiendan Sagrada? // estadísticas\n\n
Ejemplo 3:\n\n
¿Cómo se eligen los dados al inicio del juego?//informacion"
```

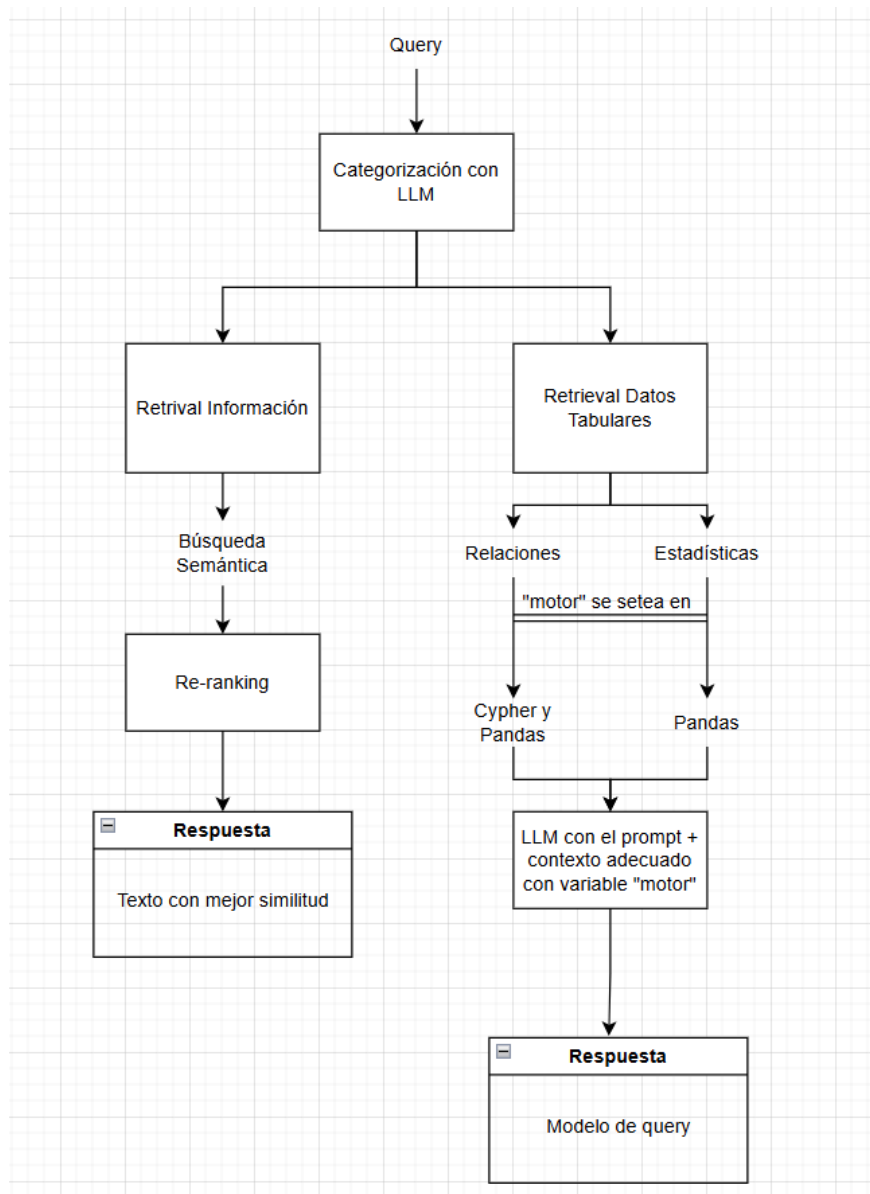
La segunda resultó más efectiva, y es con la cual decidí quedarme a la hora de pasarle al modelo. Dentro del .pynb incluí los resultados de las comparativas de las clasificaciones. En resumen, al modelo del llm con el few-shot solamente le costó un intento para clasificar correctamente 4 preguntas que no estaban en el dataset con el cual fue entrenado el modelo de redes neuronales.

LLM - FEW SHOT PROMPTING	MODELO ENTRENADO
<p>Aquí están las clasificaciones de las preguntas:</p> <ul style="list-style-type: none"> * ¿Con cuántos dados se juega? // información * ¿Cuál es el récord máximo en Sagrada? // estadísticas * ¿Quiénes pueden jugar Sagrada? // información * ¿Qué distribuidora distribuye el juego en Argentina? // relaciones 	<p>Batches: 100% 1/1 [00:00<00:00, 2.65W/s]</p> <p>1/1 0s 79ms/step</p> <p>Pregunta: ¿Con cuántos dados se juega?, Clase predicha: informacion</p> <p>Pregunta: ¿Cuál es el récord máximo en Sagrada?, Clase predicha: estadísticas</p> <p>Pregunta: ¿Quiénes pueden jugar Sagrada?, Clase predicha: relaciones</p> <p>Pregunta: ¿Qué distribuidora distribuye el juego en Argentina?, Clase predicha: relaciones</p>

En conclusión, el LLM es la mejor opción para el clasificador de categorías. Requiere menos datos de entrenamiento, es eficiente y eficaz.

Definición de un Pipeline de Recuperación

La arquitectura del pipeline quedó definida de la siguiente manera:



Categorización con LLM: En esta etapa me apoyo en el clasificador entrenado con few-prompt en el ejercicio anterior e integro la interfaz dentro del pipeline.

Retrieval de información: Se implementó la búsqueda híbrida: la comprensión semántica de los LLMs + el poder del *keyword matching* para precisión textual. En la BBDD vectorial obtengo los primeros 15 textos más cercanos, extraigo los contenidos de cada uno y creo un índice de tipo BM25 (ranking basado en palabras clave) para el re-ranking de los textos obtenidos, ejecuto la consulta sobre el índice y así obtengo el mejor documento con coincidencia léxica entre aquellos que fueron los mejores en la búsqueda semántica. El parámetro *view* fue agregado para observar mejor texto y probar que la consulta funcionara previo a la integración con la generación y conversación.

Retrieval de datos tabulares: En esta función tuve que implementar un condicional para que según la categoría devuelva el formato correspondiente, durante el testeo de esta parte algo que sucedía era que el llm clasificaba preguntas que debían ser respondidas con información como relaciones o estadísticas haciendo que el modelo que genera las queries fallara. Para subsanar esto agregué un manejo de errores para que corte la ejecución en caso de que ocurriera y muestre la categoría en la que clasificó erróneamente la consulta. Luego, si la pregunta fue clasificada correctamente, en la variable “motor” están definidos los “lenguajes” (Pandas y Zephyr) en los cuales el llm debe armar la consulta (template). Para esto, me apoyé en la interfaz creada en los puntos anteriores donde ya se había probado el llm que devuelve las consultas para los datos tabulares.

Integración para la Generación y Conversación:

En este punto, para la integración con la conversación tuve que hacer algunas modificaciones a las funciones definidas en el pipeline de recuperación para que el chat ahora, sea el que ejecute las consultas de los datos tabulares (aquellas que son de pandas) y pueda encontrar la información, logrando así un chatbot con RAG.

Para ello, se definieron dos funciones: **consultar_filtro_pandas_estadisticas** y **consultar_filtro_pandas_relaciones**, en ellas se evalúa la consulta generada por el llm en el pipeline de recuperación con los datos locales. Ejecuta esa query, si existe, y se le agrega como contexto al LLM. Quedando el flujo de la siguiente manera:

1. Categorización de la pregunta (LLM)
2. Búsqueda de información relevante según categoría
3. Contextualización del LLM, en caso de que la consulta ya haya sido evaluada
4. Inclusión del historial, si existe, para pasarle como contexto
5. Generación de la respuesta con RAG

Para lograr el punto 5 creé una función que integra los demás puntos: **process_query_with_rag** a la cual se le pasa el usuario del input, el historial del chat, y las claves de los datos tabulares para poder hacer las consultas adecuadas en el pipeline de recuperación, en caso de necesitarlas.

Los resultados de las ejecuciones de este ejercicio se encuentran en la sección de *Resultados*.

Ejercicio 2

En este ejercicio me apalanque de las funciones definidas en el punto anterior. Se encapsularon las funciones dentro de otras para estandarizar los nombres. En la implementación del agente ReAct, use la librería de **Langchain** que provee el sistema de agente y decidí usar el **ZERO_SHOT_REACT_DESCRIPTION** que hace “reasoning” antes de tomar una decisión, sacando la necesidad de entrenamiento y proveyendo una respuesta adecuada(4)

También, se implementaron dos funciones para la búsqueda en línea con DuckDuckGo y Wikipedia. Las fuentes de donde el agente puede consultar están definidas en una lista a la cual consulta según donde considere que hallará la respuesta para el prompt. Es decir que el LLM no sabe qué hace la función **duckduckgo_search** a nivel de código, por ejemplo, pero sí entiende la descripción en lenguaje natural: "Busca en toda la web". Esta descripción es la que le permite al LLM decidir cuándo es apropiado usar cada herramienta.

En la instancia del modelo, se dejó la temperatura por default de cero para que las respuestas del modelo sean lo más determinísticas y objetivas posible, minimizando la creatividad o la aleatoriedad.

El flujo final es el siguiente:

1. **Input:** El agente recibe la pregunta: "¿Los daltónicos pueden jugar Sagrada?".
2. **Razonamiento** (Thought): El LLM analiza la pregunta. Reconoce que "Sagrada" es probablemente un juego y "daltónicos" se refiere a la ceguera al color. Concluye que necesita información sobre las reglas y componentes del juego "Sagrada" para ver si el color es un elemento crucial.
3. **Acción** (Action): El LLM examina las descripciones de sus herramientas. Para obtener información general sobre un juego de mesa..

4. **Observación** (Observation): El agente ejecuta la herramienta elegida y accede a la información que encontró
5. **Razonamiento y síntesis**: El agente recibe este resultado (la observación). El LLM analiza esta nueva información. Si la información es suficiente, "piensa" que ya tiene todo lo necesario. A partir de los datos recopilados (por ejemplo, que el juego depende de dados con colores y símbolos, pero los símbolos ayudan a diferenciarlos), el LLM sintetiza y genera la respuesta final en lenguaje natural.

Resultados y Pruebas

En esta sección, se encuentran los resultados de los ejercicios implementados en el punto 1 y 2.

Ejercicio 1

A modo de testeo, primero se le pasó al chat que respondería sin tener los datos a consultar:

```
[ ] test_prompt = "User query: Quien diseña SagradaViu"
[ ] llm.predict(test_prompt)

22 /opt/python-lpqr-89-4223887933.py:1: LangChainDeprecationWarning: The method 'BaseChatModel.predict' was deprecated in langchain-core 0.1.7 and will be removed in 1.0. Use 'method="invoke"' instead.
  llm.predict(test_prompt)
'El arquitecto más conocido y el que le dio su forma distintiva a la "Sagrada familia" es "Antoni Gaudí". Vivia en Barcelona, es importante saber que el diseño original no fue suyo. La primera piedra se colocó en 1882 bajo la dirección del arquitecto "Francisco de Paula del Villar y Lozano". Villaverd renunció poco después, y en 1883, "Antoni Gaudí" asumió el proyecto. Gaudí transformó completamente el diseño original, dedicando los últimos años de su vida exclusivamente a la Sagrada familia y diseñando la villa y el estilo modernista que hoy conocemos. Desde la muerte de Gaudí en 1926, varios arquitectos han continuado su obra, siguiendo sus planos y maquetas, aunque la interpretación de su visión ha sido un tema de debate.'
```

da una respuesta trivial y con información con la cual fue seguramente entrenado. Como es de esperar, aún no relaciona la palabra “Sagrada” con el juego de mesa sino con la Iglesia católica la “Sagrada Familia”.

Ahora bien, instanciamos nuevamente el chat y usamos el pipeline de retrieval para que consulte la información antes de formular la respuesta usando la función

process_query_with_rag:

```

print("Conversación iniciada. Escribe 'exit' para salir.")

while True:
    user_input = input()
    if user_input.lower() == "exit":
        break
    respuesta = process_query_with_llm(user_input, memory, resume)
    print(f"Asistente: {respuesta}")

Conversación iniciada. Escribe 'exit' para salir.

Cuántos comentarios tiene Sagrada?
La categoría es: estadísticas
Query a ejecutar: df_estadisticas[df_estadisticas['Category'] == 'GAME STATS'] & (df_estadisticas['Metric'] == 'Comments')
Asistente: Sagrada tiene 5137 comentarios.

Los niños pueden jugar?
La categoría es: Información
Asistente: No se encontró información relacionada con si los niños pueden jugar en el texto proporcionado. El texto se centra en las reglas de puntuación y el modo solitario del juego.

Can kids play?
La categoría es: Información
Asistente: Sí, los niños pueden jugar. El texto menciona que es un juego muy accesible y que "cualquiera podría jugarlo". Específicamente, se dice que "un niño de ocho años que obviamente sepa leer podría jugar a este juego". También se destaca que es una gran oportunidad para enseñar a los niños las sutilezas del juego.

Cuándo termina la partida?
La categoría es: Información
Asistente: No se encontró información relacionada con cuándo termina la partida en la información proporcionada.

Cuándo termina el juego?
La categoría es: Información
Asistente: El juego termina al final de la ronda 10.

Cuántos jugadores pueden jugar?
La categoría es: estadísticas
Query a ejecutar: df_estadisticas[df_estadisticas['num_players'] == 4]
Asistente: No se encontró información sobre cuántos jugadores pueden jugar. La información proporcionada indica un error al evaluar el número de jugadores.

de a cuantos jugadores se puede jugar?
La categoría es: Información
Asistente: Según la información proporcionada, el juego se puede jugar en "Modo Solitario", lo que significa que se puede jugar con "un jugador".

No se encontró información sobre el número de jugadores para el modo no solitario.

Quiénes diseñaron sagrada?
La categoría es: relaciones
Asistente: No se encontró información relacionada con quién diseñó la Sagrada familia.

quiénes crearon sagrada?
La categoría es: relaciones
Asistente: Los creadores de Sagrada son Adrian Adamescu, Daryl Andrews y Peter Macken.

exit

```

El chat logra consultar los datos de forma dinámica y responder con esa información. Una limitación es que el clasificador puede cometer errores y clasificar erróneamente las prompts haciendo que el chat falle en caso de que la query de Pandas no funcione porque debería haber sido clasificada como información, pero en caso de que haya sido clasificada bien y dicha información no esté en la fuente de datos entonces le solicita al usuario que vuelva a consultarle:

```

Quiénes diseñaron sagrada
La categoría es: Información
Asistente: No se encontró nada relacionado con quiénes diseñaron el juego "Sagrada" en la información proporcionada. Por favor, reformula la pregunta si buscas otra cosa.

```

En la imagen de arriba, se le pregunta al chatbot *quienes diseñaron Sagrada?* Una pregunta que debió haber sido clasificada como “relaciones” el clasificador la clasificó como “información” esto hace que el chatbot no encuentre información relacionada y le diga al usuario que vuelva a reformular la pregunta.

En cuanto al idioma, a pesar de tener fuentes de datos en distintos idiomas, el chat logra identificar los textos correctamente para la búsqueda de la información. Lo probé con preguntas en inglés y en español. Sin embargo, una dificultad del chat es que si el contexto que encontró

está en inglés, entonces el chat no logra interpretar la consulta, pero al pasarle la traducción de la query, logra responder y en español:

```

Los niños pueden jugar?
La categoría es: informacion
Asistente: No se encontró información relacionada con si los niños pueden jugar en el texto proporcionado. El texto se centra en las reglas de puntuación y el modo solitario del juego.

Can kids play?
La categoría es: informacion
Asistente: Sí, los niños pueden jugar. El texto menciona que es un juego muy accesible y que "cualquiera podría jugarlo". Específicamente, se dice que "un niño de ocho años que obviamente sepa leer podría jugar a este juego". También se destaca que es una gran oportunidad para enseñar a los niños las sutilezas del

```

Otra limitación que se le presenta, es en las relaciones. Si el verbo usado en la pregunta no está explícitamente como “relación” en el dataframe, entonces el chat no puede hallar los datos para responder:

```

quienes son los publishers de Sagrada
La categoría es: relaciones
Asistente: Los publishers de Sagrada son:

```

```

* Floodgate Games
* Cranio Creations
* CrowD Games
* Devir
* Engames
* FoxGames
* Galápagos Jogos
* Geekach Games
* Lanlalen
* Lex Games
* Mandoo Games
* Matagot
* MEBO Games
* MINDOK
* MIPL
* One Moment Games
* Pegasus Spiele
* Reflexshop
* Yes Papa Games

```

```

Quienes publicaron sagrada?
La categoría es: informacion
Asistente: No se encontró información sobre quién publicó Sagrada en el texto proporcionado.

```

En el ejemplo de arriba cuando se le pregunta explícitamente con la palabra “publishers” entonces si logra traer la información; pero, al preguntarle sin la palabra en esta ocasión clasificó erróneamente la pregunta era de “relaciones” y la clasificó como “información” esto podría ser

derivado de que dependiendo de cómo se estructure la información la respuesta a esta pregunta podría hallarse en los textos en lugar de los dataframes.

```
Quienes diseñaron sagrada?
La categoría es: relaciones
Asistente: No se encontró información relacionada con quién diseñó la Sagrada Familia.

quienes crearon sagrada?
La categoría es: relaciones
Asistente: Los creadores de Sagrada son Adrian Adamescu, Daryl Andrews y Peter Wocken.
```

Para las preguntas clasificadas como estadísticas, si encuentra dicha información logra traerla correctamente:

```
Cuantos comentarios tiene Sagrada?
La categoría es: estadísticas
Query a ejecutar: df_estadísticas[(df_estadísticas['Category'] == 'GAME STATS') & (df_estadísticas['Metric'] == 'Comments')]
Asistente: Sagrada tiene 6137 comentarios.
```

Ejercicio 2

El agente ReAct con **ZERO_SHOT_REACT_DESCRIPTION** muestra todo el razonamiento que está haciendo previo a responder la consulta, todo esto lo hace en inglés, sin embargo, logra responder en el mismo idioma de la query:

```
> Entering the AgenticActor chain...
Action: wikipedia_search
Action Input: 'sagrada board game'
The code that caused this warning is on line 380 of the file /usr/local/lib/python3.11/dist-packages/wikipedia/wikipedia.py. To get rid of this warning, pass the additional argument 'features="lxml"' to the BeautifulSoup constructor.
lls ~ BeautifulSoup(html).find_all('li')

Observation: Page: Sagrada (board game)
Summary: Sagrada is a dice-drafting board game designed by Adrian Adamescu and Daryl Andrews and published in 2017 by Floodgate Games. Each player constructs a stained-glass window using dice on a personal 4x5 game board with restrictions on the types of dice that can be played on each space. Players gain points by using the rules above only one die can be placed in the middle four space here and that is a red four we draft a red three but this lets us use one of the available tool cards we pay one tool token to use the card and change the three to a four subsequent uses of a tool card would use two charges If you ever accidentally break the playing rules you must remove those dice once 10 rounds are up scoring takes place you score the points shown in the bottom left of the goal for each time you achieve that goal as in this example we score 11 twice your private goal will always be the total value of one color and finally lose one point for each empty square why would you like this game? Sagrada is a puzzle game for people who love spatial puzzles, and the rising tension of having your available placement options get more and more limited as the game goes on and that's the best part of Sagrada looking at the draft options and realizing that you have put yourself in a bad position or the triumph of getting a row exactly right to match a goal The tool cards give you different actions on how to manipulate the game and only three are included each time you play and well it's a pretty looking game those shiny dice are really nice the best thing about this game is when it all works well and you have a wonderful looking high scoring completed window however, despite the potential of tools, tiles and goals in the game if you don't like the core gameplay of drafting and placement none of the changes will make the game work for you there's also limited interest while drafting to hurt other players can happen you're far more likely to draft for your own needs first like the mechanics but after the game has ended and you try roll player and for a different type of pretty dice rafting game try noctiluca sagrada one pretty puzzle If you enjoyed this video hit the notification button subscribe to the channel and come support us on patreon Thought/Insight: The question asks if colored/die people can play Sagrada. From the 'wikipedia_search' result, Sagrada is a dice-drafting game where players construct a stained-glass window using dice, with restrictions on placement. From the 'doc_search' result, I learned that dice have both colors and numbers (e.g., "red four", "red three"). Crucially, the document states: "your private goal will always be the total value of one color". This means that a player's personal scoring objective depends entirely on being able to identify and sum die of one color. Final Answer: Sería difícil para las personas daltónicas jugar Sagrada de manera efectiva, ya que un objetivo privado clave en el juego es sumar el valor total de los dados de un color específico. Si un jugador no puede distinguir los colores, no podrá perseguir este objetivo crucial. > Finished chain. Sería difícil para las personas daltónicas jugar Sagrada de manera efectiva, ya que un objetivo privado clave en el juego es sumar el valor total de los dados de un color específico. Si un jugador no puede distinguir los colores, no podrá perseguir este objetivo crucial.
```

Dependiendo de la fuente que se use puede dar respuestas redactadas de diferente manera pero

con información acertada, en el ejemplo de arriba se le pasó la misma pregunta pero decidió responder con Wikipedia, mientras que en la de abajo decidió hacer una búsqueda en DuckDuckgo. Ambas fueron respondidas correctamente, la segunda fue menos costosa en términos de “razonamiento”:



```
respuesta = agent.run("¿En que año se publicó Sagrada?")
print(respuesta)
```

> Entering new AgentExecutor chain...

Action: duckduckgo_search

Action Input: Sagrada board game publication year

Observation: *Game of the Year (abbreviated GotY) is a title awarded annually by various magazines, websites, and game critics to deserving tabletop games, including board games.*

Thought: I now know the final answer

Final Answer: Sagrada se publicó en 2017.

> Finished chain.

Sagrada se publicó en 2017.

Conclusiones

Ambas implementaciones, Chatbot con arquitectura RAG y Agente ReAct demostraron ser funcionales. Me permitieron interactuar con un corpus de conocimiento heterogéneo incluyendo datos textuales, tabulares y de grafos.

En el chatbot RAG, la performance quedó atada al desempeño del llm que actúa como clasificador. A pesar de haber tenido la mejor performance, puede aún cometer errores a la hora de clasificar las queries, sin embargo, esto valida el enorme poder que tienen los LLMs para realizar tareas de clasificación con muy pocos datos. El pipeline de recuperación forma una parte fundamental a la hora de proveer la información correcta al contexto del chatbot y que éste dependa del clasificador como primer paso lo deja expuesto a la performance del clasificador avanzado, como mejora a futuro se podría implementar una especie de “enrutador” para que pueda redirigir la consulta a múltiples categorías a la vez.

Por otro lado, el agente ReAct que implementado en el punto 2 demostró una capacidad de razonamiento dinámico superior. Al darle la capacidad de elegir autónomamente entre las herramientas proporcionadas y búsquedas web externas, el agente mostró mayor resiliencia y versatilidad. Fue capaz de responder preguntas que estaban fuera del alcance de mi base de datos local. En esta implementación no se depende en absoluto del clasificador ya que el chat tiene la capacidad de razonar y evaluar si tiene todo lo necesario para su respuesta.

En conclusión ambas implementaciones requieren una cuidadosa integración de cada parte (búsquedas), un manejo de datos robustos y una interpretación de los resultados del chat. En mi opinión no creo que una sea mejor que la otra, sino que la elección depende de un balance entre control versus flexibilidad. En cuanto a la implementación es más directa la del ReAct, porque le delega gran parte del razonamiento y la orquestación de herramientas al propio LLM. Sin embargo si lo que se quiere preservar y priorizar es el acceso a la información para las respuestas entonces la arquitectura RAG podría ser más óptima aunque es más compleja de ensamblar pieza por pieza, garantiza que el modelo se ancle exclusivamente a los datos verificados que le proporcionamos. Esto minimiza drásticamente el riesgo de alucinaciones o respuestas basadas en conocimiento externo no validado, ofreciendo un sistema con mayor previsibilidad y control sobre el resultado final.

Referencias

Sentence-transformers. (s.f.). *all-mpnet-base-v2*. Hugging Face. Consultado el 29 de junio de

2025, de <https://huggingface.co/sentence-transformers/all-mpnet-base-v2>

Pinecone. (s.f.). *LangChain*. Pinecone Docs. Consultado el 29 de junio de 2025, de

<https://docs.pinecone.io/integrations/langchain>

CasadoPedro. (2023). *TP1_NLP_P2* [Repositorio de GitHub]. GitHub.

https://github.com/CasadoPedro/TP1_NLP_P2

Ewoudvdw. (2023, 11 de octubre). [*Feature Request*] *It would be great to have built-in support*

for agents to use tools in parallel [Discusión en línea]. GitHub.

<https://github.com/langchain-ai/langchain/discussions/12888>