



Tecnicatura Universitaria en Inteligencia Artificial

Trabajo Práctico Final NLP - Sagrada

Autor: Caballero Franco

Asignatura: Procesamiento del Lenguaje Natural

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

Fecha de entrega: 29/06/2025

ÍNDICE

1. Resumen	pág. 2
2. Introducción	pág. 3
3. Metodología	pág. 3-5
4. Desarrollo	pág. 5-7
5. Resultados	pág. 7-10
6. Conclusiones	pág. 10-11
7. Referencias	pág. 11-12
8. Anexos	pág. 12

RESUMEN

En el presente trabajo práctico se desarrolló un sistema conversacional experto basado en la arquitectura RAG (Retrieval-Augmented Generation) y su posterior evolución hacia un agente autónomo capaz de seleccionar herramientas según la consulta del usuario. El proyecto fue implementado utilizando como caso de estudio el juego de mesa asignado, integrando distintas fuentes de información: documentos textuales, estadísticas tabulares y relaciones estructuradas en una base de datos de grafos.

Para la primera etapa, se construyeron tres módulos de recuperación independientes: uno basado en búsqueda semántica híbrida sobre una base vectorial, otro para consultas dinámicas sobre tablas estadísticas y un tercero orientado a la recuperación de conocimiento desde un grafo consultable mediante lenguaje Cypher. Se diseñaron interfaces específicas para cada fuente de datos y se implementó un pipeline de recuperación que incluye clasificación de intención y generación de consultas estructuradas.

En la segunda etapa, se implementó un agente autónomo utilizando LangChain, bajo el paradigma ReAct, integrando las herramientas desarrolladas anteriormente junto con opciones de búsqueda online. Se configuró un prompt de sistema que guía el razonamiento y la elección de herramientas por parte del agente.

Se realizaron múltiples pruebas de funcionamiento para evaluar la capacidad de recuperación y generación del sistema en distintos escenarios. Finalmente, se analizan los resultados obtenidos, las limitaciones detectadas y se presentan recomendaciones para mejorar el desempeño general del agente.

INTRODUCCIÓN

El crecimiento exponencial de la información disponible en línea ha impulsado la necesidad de desarrollar sistemas capaces de recuperar y sintetizar conocimiento de manera eficiente. En este contexto, las arquitecturas basadas en Retrieval-Augmented Generation (RAG) han demostrado ser una solución efectiva, combinando recuperación de información con generación de lenguaje natural para proporcionar respuestas contextualizadas y precisas.

Este trabajo práctico se enmarca dentro de esa tendencia tecnológica, teniendo como objetivo construir un chatbot experto en el juego de mesa asignado, utilizando la arquitectura RAG como base. La implementación consideró tres fuentes heterogéneas de datos: documentos textuales, estadísticas tabulares y relaciones estructuradas en una base de datos de grafos. A partir de estas fuentes, se desarrollaron mecanismos de búsqueda y recuperación específicos, integrando técnicas de NLP, recuperación semántica híbrida y consultas estructuradas en Cypher.

Como evolución del sistema RAG, se implementó un agente autónomo basado en el paradigma ReAct, utilizando LangChain. Este agente es capaz de seleccionar, de forma razonada, la herramienta más adecuada para cada consulta, integrando además fuentes de información externas como Wikipedia y DuckDuckGo para ampliar su capacidad de respuesta.

El trabajo busca no solo aplicar los conceptos teóricos abordados durante el curso, sino también poner en práctica técnicas avanzadas de NLP, recuperación de información y generación de lenguaje. La estructura del informe se organiza en las siguientes secciones: metodología, donde se detallan las decisiones técnicas y herramientas utilizadas; desarrollo e implementación, donde se describe el proceso de construcción del sistema; resultados, donde se presentan las pruebas realizadas; y finalmente, las conclusiones, que incluyen un análisis crítico del desempeño y posibles mejoras futuras.

METODOLOGÍA

Fuentes de datos:

Se utilizaron las tres fuentes de datos previamente generadas en el Trabajo Práctico 1:

- Información: Conjunto de textos extensos relacionados con el juego de mesa Sagrada, extraídos mediante Web Scraping del sitio web [MisutMeeple - Sagrada](#). Estos textos fueron previamente organizados y almacenados en formato .txt dentro del repositorio del proyecto.
- Estadísticas: Un dataset tabular conteniendo datos estructurados del juego, como rankings, puntajes, número de jugadores, entre otros indicadores relevantes. Estos

fueron extraídos mediante web scraping del sitio web [Board Game Geek - Sagrada](#)

- Relaciones: Un conjunto de relaciones entre entidades del juego, tales como autores, editores, diseñadores, artistas, etc. destinadas a ser cargadas en **una base de datos de grafos**. Estos también fueron extraídos mediante web scraping del sitio web [Board Game Geek - Sagrada](#)

Herramientas y Tecnologías usadas:

El desarrollo del sistema se realizó utilizando el lenguaje de programación Python, complementado por las siguientes tecnologías y librerías:

- **LangChain**: Para la implementación del agente autónomo y la orquestación de herramientas.
- **Sentence-Transformers**: Para la generación de embeddings semánticos.
- **ChromaDB**: Para la construcción de la base de datos vectorial.
- **Pandas**: Para el manejo y filtrado de datos tabulares.
- **Neo4j**: Como motor de base de datos de grafos, consultable mediante Cypher.
- **Scikit-learn / Transformers**: Para el entrenamiento del clasificador de intención y el uso del LLM generador.
- **DuckDuckGoSearchRun / WikipediaAPIWrapper**: Para búsquedas online en la fase del agente autónomo.

Decisiones Clave de Diseño:

Modelo de Embedding

Se seleccionó el modelo **all-MiniLM-L6-v2** de Sentence-Transformers por su balance entre velocidad de inferencia, bajo consumo de recursos y buena capacidad de representación semántica. Este modelo fue entrenado específicamente para tareas de similitud y recuperación semántica, lo que lo hace adecuado para la búsqueda en bases vectoriales.

Text Splitter

Se utilizó un **TextSplitter tipo RecursiveCharacterTextSplitter**, ajustando la longitud máxima de los chunks y el solapamiento entre ellos para optimizar la cobertura semántica y evitar cortes abruptos en el contenido.

Motor de Base Vectorial

Para la base de datos vectorial, se optó por **ChromaDB**, considerando su velocidad de búsqueda, fácil integración con LangChain y buen rendimiento en consultas semánticas de baja latencia.

Clasificador de Intención

Se compararon dos aproximaciones:

- Un **modelo de clasificación tradicional** (entrenado previamente en el TP1) utilizando técnicas de NLP y Machine Learning supervisado.
- Un **clasificador basado en LLM**, el modelo *Gemini 2.5 Flash*, utilizando Few-Shot Prompting.

Tras comparar el modelo entrenado del TP1 con un clasificador LLM basado en few-shot prompting, se observó un mejor desempeño del segundo en métricas como **accuracy** y **macro-F1**. Por ello, se seleccionó el modelo LLM para el enrutamiento de consultas.

Motor de Grafos

Se implementó una base de datos **Neo4j** para almacenar las relaciones extraídas, permitiendo consultas mediante el lenguaje **Cypher**.

DESARROLLO

Ejercicio 1 – Arquitectura RAG

Se construyeron tres interfaces independientes para la recuperación de información desde cada fuente:

- **Interfaz para documentos:**
Se implementó una búsqueda híbrida combinando BM25 para la parte léxica y similitud de embeddings para la parte semántica. Posteriormente, se aplicó un Re-Rank utilizando el propio modelo de embeddings para mejorar la calidad de los fragmentos recuperados.
- **Interfaz para datos tabulares:**
Se cargaron los datos en un DataFrame de Pandas. Se calcularon estadísticas globales (valores únicos, máximos, mínimos y promedios) de cada campo. Se diseñó un sistema donde, ante una consulta, un LLM genera filtros que luego se aplican directamente sobre el DataFrame para obtener la respuesta.
- **Interfaz para datos de grafos:**
Se insertaron las relaciones en una base Neo4j y se implementó un sistema para que el LLM traduzca consultas en lenguaje natural a queries Cypher. Esto permitió obtener respuestas específicas a partir de relaciones entre entidades.

Clasificador de Intención

Como se mencionó antes, se comparó el modelo entrenado en el TP1 y el clasificador LLM basado en few-shot prompting, este último fue quien tuvo mejor desempeño

Pipeline Final del RAG

Se integraron todos los componentes anteriores bajo un pipeline conversacional. Este flujo permite recibir una consulta, identificar su intención, recuperar la información desde la fuente correcta, y generar una respuesta con el LLM, respetando el idioma original del usuario.

Ejercicio 2 – Agente Autónomo ReAct

Se implementó un agente autónomo utilizando **LangChain** bajo el paradigma **ReAct (Reason + Act)**.

Se encapsularon las siguientes herramientas como funciones independientes dentro del agente:

- `doc_search()`: Recuperación en base vectorial con re-rank.
- `table_search()`: Consultas dinámicas a datos tabulares.
- `graph_search()`: Consultas estructuradas al grafo mediante Cypher.
- `wikipedia_search()`: Búsqueda en Wikipedia.
- `duckduckgo_search()`: Búsqueda abierta en internet.

Prompt de Sistema del Agente

Se construyó un prompt que describe el propósito y funcionamiento de cada herramienta, así como ejemplos de razonamiento paso a paso (Thought) y selección de acciones (Action). Esto permitió que el agente pueda decidir de manera autónoma cuál o cuáles herramientas utilizar según la complejidad y la naturaleza de cada consulta.

Proceso de Ejecución

Ante cada consulta, el agente realiza el siguiente ciclo:

1. Analiza la consulta y decide cuál herramienta ejecutar.

2. Ejecuta la herramienta seleccionada.
3. Evalúa la respuesta parcial obtenida.
4. Si es necesario, realiza una segunda acción o combina resultados.
5. Genera la respuesta final para el usuario.

Pruebas Realizadas

Se realizaron al menos cinco pruebas distintas para el RAG y cinco para el agente autónomo, cubriendo casos como:

- Preguntas sobre reglas del juego.
- Consultas estadísticas específicas.
- Búsquedas de relaciones entre entidades.
- Consultas ambiguas que requirieron múltiples herramientas.
- Preguntas sin respuesta directa en el corpus.

RESULTADOS

Nota sobre los outputs mostrados:

Algunos de los resultados presentados en este informe corresponden a outputs generados en notebooks o entornos de desarrollo cuyo formato original es scrollable. Por cuestiones de espacio y formato del documento, solo se incluyen capturas parciales de estos outputs.

El código completo y todos los resultados detallados están disponibles en el anexo, en la carpeta de código fuente.

Ejercicio 1 – Sistema RAG Integrado

Se realizaron múltiples pruebas sobre el sistema RAG desarrollado, cubriendo consultas que involucran cada una de las fuentes de datos (documentos, estadísticas y relaciones). A continuación, se detallan algunos ejemplos representativos:

```
Asistente: Hola, ¿en qué puedo ayudarte?  
Consulta: Quienes son los diseñadores de Sagrada?  
Asistente: Los diseñadores de Sagrada son Daryl Andrews y Adrian Adamescu.
```

Consulta: Como se juega?

Asistente: Sagrada se juega a lo largo de 10 rondas.

****Cada ronda se desarrolla de la siguiente manera:****

- **Lanzamiento de Dados**:** El jugador inicial de la ronda saca una cantidad de dados de la bolsa (5 para 2 jugadores, 7 para 3, 9 para 4).
- **Turnos**:**
 - Los jugadores se turnan dos veces cada uno en una secuencia específica: primero, en sentido horario desde el jugador inicial, cada uno saca un dado.
 - **En su turno, un jugador puede realizar una o ambas de estas acciones:****
 - **Seleccionar y Colocar un Dado**:** Tomar 1 dado del Draft Pool y colocarlo en un espacio abierto de su tablero de ventana. La colocación debe seguir reglas de adyacencia y color.
 - **Usar una Carta de Herramienta**:** Pagar Tokens de Favor (1 si la carta está libre, 2 si ya tiene tokens) para usar la habilidad de la carta.
- **Fin de la Ronda**:** Todos los dados restantes en el Draft Pool se colocan en el marcador de Ronda. La bolsa de dados y el rol de jugador inicial se rotan.

El juego concluye al final de la décima ronda.

Consulta: Como funcionan los dados?

Asistente: Los dados en Sagrada funcionan como representaciones de piezas de vidrio para vitrales. Tienen un color (verde, amarillo, rojo, azul) y un valor (1-6).

Su funcionamiento en el juego es el siguiente:

- **Creación del Draft Pool**:** Al inicio de cada ronda, se sacan dados de una bolsa (una cantidad específica según el número de jugadores).
- **Selección y Colocación**:** Los jugadores seleccionan un dado de este Draft Pool para colocarlo en un espacio abierto de su ventana, siguiendo reglas de adyacencia y color.
- **Uso de Herramientas (Modo Solitario)**:** En el modo solitario, los dados del Draft Pool también pueden ser gastados para activar Cartas de Herramienta.
- **Fin de Ronda**:** Los dados restantes del Draft Pool se colocan en el Marcador de Ronda.
- **Puntuación**:** Los valores de los dados de colores específicos en la ventana contribuyen a los objetivos privados. En modo solitario, el jugador también gana puntos por completar objetivos públicos.

Consulta: Cuantos comentarios tiene Sagrada?

Asistente: Sagrada tiene 19 comentarios.

Consulta: Que dicen algunos de ellos?

Asistente: Según la transcripción de la reseña, ****Edo**** dice que el tren del hype por Sagrada está "fuera de control". Lo describe como un juego "más grande" de lo que esperaba, pero "ligero y etéreo", con una interacción entre jugadores "bastante liviana".

Lo clasifica como un juego "ligero, accesible, de entrada" que "pasa rápido" y lo compara con "Herbaceous" por ser fácil de explicar y por tener una mecánica de cartas que le da profundidad.

Ejercicio 2 – Agente Autónomo ReAct

Se realizaron cinco pruebas adicionales donde el agente autónomo debía decidir qué herramienta utilizar (documentos, estadísticas, relaciones, wikipedia, duckgo).

Query: “Como se juega a Sagrada?”

```
Thought: The user is asking for instructions on how to play "Sagrada".
Action:
json
{
  "action": "Document Search",
  "action_input": "Como se juega Sagrada"
}
```

Sagrada es un juego de dados y ventanas de colores donde los jugadores son artesanos que construyen vidrieras.

****El objetivo**** es ganar la mayor cantidad de puntos de victoria (PV) al final de 10 rondas.

****Preparación**:**

- Cada jugador elige un tablero de ventana (con un patrón de vidriera y nivel de dificultad) y fichas de favor según la dificultad.
- Se reparte una carta de Objetivo Privado a cada jugador.
- Se revelan 3 cartas de Objetivo Público y 3 cartas de Herramienta en el centro de la mesa.
- Todos los dados de colores se meten en una bolsa.

Query: "Cuál es el rating de Sagrada y cuanta gente lo ha jugado?"

Thought: The user is asking for the rating and the number of people who have played "Sagrada".

```
Action:
json
{
  "action": "Table Search",
  "action_input": "rating and number of players for Sagrada"
}
```

```
> Finished chain.
Sagrada tiene un rating de 7.9/10 y ha sido jugado por 85,321 personas.
```

Query: "Listame los artistas que están relacionados con el juego Sagrada y contame un poco sobre alguno de ellos"

Thought: The user is asking for artists related to the game "Sagrada" and then to provide information about one of them. I should start by using `Graph Search` to find entities related to "Sagrada", specifically looking for artists or creators.

Action:

```
json
{
  "action": "Graph Search",
  "action_input": "artistas relacionados con el juego Sagrada"
}
```

Observation: Sagrada está relacionado con los artistas Adrian Adamescu y Peter Wocken.

Thought: I have found two artists: Adrian Adamescu and Peter Wocken. Now I need to choose one and get more information about them.

Action:

```
json
{
  "action": "Wikipedia Search",
  "action_input": "Adrian Adamescu artista"
}
```

```
> Finished chain.
Los artistas relacionados con el juego Sagrada son Adrian Adamescu y Peter Wocken.
```

Adrian Adamescu es un artista e ilustrador rumano, reconocido por su trabajo en la industria del diseño de juegos de mesa y videojuegos.

Query: "Basandonos en las reglas de Sagrada ¿Qué otros juegos que son parecidos existen?"

Thought: The user is asking for games similar to "Sagrada" based on its rules.

```
Action:
json
{
  "action": "Document Search",
  "action_input": "Reglas de Sagrada"
}
```

```

Observation: "Sagrada es un juego de mesa de construcción de vidrieras. Los jugadores tiran dados de colores y los colocan en su tablero per
Thought: I have a good understanding of Sagrada's core mechanics: dice drafting, pattern building on a personal board, color and number rest

Action:
json
{
  "action": "DuckDuckGo Search",
  "action_input": "juegos de mesa con draft de dados y construcción de patrones"
}

Observation: "Resultados de búsqueda sugieren juegos como: Azul (draft de losetas, patrón de colores), Roll for the Galaxy (draft de dados y
Thought: The DuckDuckGo search provided several games. I need to filter them based on how similar they are to Sagrada's core mechanics: dice
...

Basándonos en las reglas de Sagrada, que incluyen el draft de dados, la construcción de patrones en un tablero personal con restricciones de

1. Azul: Aunque usa losetas en lugar de dados, comparte la mecánica central de "drafting" (selección de componentes de un área común) y
2. Ganz Schön Clever: Es un juego "roll-and-write" que se enfoca en la manipulación de dados para completar patrones numéricos y de col

```

Query: "Como funcionan las cartas en Sagrada?"

```

Thought: The user is asking about how cards work in the game "Sagrada".
Action:
json
{
  "action": "Document Search",
  "action_input": "cómo funcionan las cartas en Sagrada"
}

```

```

En Sagrada, las cartas funcionan de varias maneras:
1. Cartas de Patrón (Vidrieras): Son el tablero individual de cada jugador y muestran un patrón de vidriera a completar. Tienen requis
2. Cartas de Herramienta: Proporcionan habilidades especiales que los jugadores pueden usar durante la partida. Estas habilidades puede
3. Cartas de Objetivo Público: Son visibles para todos los jugadores y otorgan puntos de victoria al final de la partida si se cumplen
4. Cartas de Objetivo Privado: Cada jugador recibe una en secreto al inicio de la partida. Otorga puntos de victoria adicionales al fir

```

CONCLUSIONES

A lo largo de este trabajo práctico se implementó un sistema completo de Recuperación Aumentada con Generación (RAG), aplicado a un dominio específico "Sagrada", utilizando múltiples fuentes de información previamente estructuradas: documentos de texto, datos tabulares y relaciones entre entidades. Se integraron técnicas de NLP, embeddings semánticos, generación de queries automáticas y clasificación de intención, con el objetivo de construir un chatbot conversacional capaz de responder preguntas complejas de forma precisa.

Se logró construir e integrar satisfactoriamente un **clasificador de intención**, tres **interfaces independientes de recuperación** (documental, tabular, grafo) y un **modelo generativo con memoria conversacional**, respetando las restricciones de cada tipo de fuente (sin pasar todos los datos al contexto).

En la segunda parte del trabajo, se desarrolló un agente autónomo ReAct mediante **LangChain**, capaz de seleccionar de manera dinámica cuál herramienta utilizar según el tipo de consulta. En las pruebas realizadas, el agente demostró una capacidad aceptable para seleccionar la herramienta adecuada y combinar fuentes de forma efectiva.

En general, se cumplieron los objetivos del trabajo, mostrando una implementación funcional del paradigma RAG con extensión a agentes autónomos, aplicando conceptos de IA, NLP y sistemas conversacionales.

Mejoras Propuestas y Fallos Detectados

Durante las pruebas y el desarrollo, se identificaron algunos aspectos que podrían mejorarse para aumentar la robustez y precisión del sistema:

- **Consultas Cypher con errores sintácticos:** En algunos casos, las consultas generadas por el LLM hacia la base de grafos contenían errores de sintaxis o referencias incorrectas. Esto podría mitigarse incorporando validaciones automáticas o prompts más estructurados.
- **Elección incorrecta de herramienta en consultas ambiguas:** Ante preguntas vagas o muy generales, el agente no siempre seleccionaba la herramienta más adecuada. Una solución sería refinar el prompt del sistema para reforzar la identificación de la intención, o implementar un modelo de respaldo para la desambiguación previa.
- **Generación de respuestas sin fundamento (“alucinaciones”):** Como suele ocurrir con modelos generativos, hubo casos en que el chatbot elaboró respuestas plausibles pero sin evidencia en los datos. Se sugiere reforzar la instrucción al modelo para que evite responder sin suficiente información y promueva respuestas del tipo: “No tengo información suficiente para responder eso”.
- **Rendimiento del clasificador tradicional vs LLM:** Aunque ambos clasificadores funcionaron correctamente, el clasificador basado en LLM mostró mejor generalización ante consultas no vistas, pero con un mayor tiempo de respuesta. Una posible mejora futura sería un sistema híbrido que utilice el clasificador tradicional como filtro rápido inicial y recurra al LLM solo en casos ambiguos.
- **Tendencia a priorizar fuentes externas:** Aunque el agente funcionó bien en muchos casos, se observó una tendencia a preferir herramientas externas (como Wikipedia o DuckDuckGo) incluso cuando la información podía recuperarse desde las fuentes internas estructuradas. Esta conducta afecta la trazabilidad y justificación de las respuestas, ya que el contenido externo no siempre puede verificarse con el mismo rigor.

REFERENCIAS

Hugging Face, Inc. (2020). Transformers: State-of-the-art Natural Language Processing for Pytorch and TensorFlow 2.0. <https://huggingface.co/transformers/>

SpaCy, (2025). SpaCy Documentation. <https://spacy.io/>

Sentence-Transformers, (2025). Sentence-Transformers: State-of-the-art Sentence Embeddings. <https://www.sbert.net/>

Neo4j. (2024). Neo4j Cypher Manual. <https://neo4j.com/docs/cypher-manual/current/>

Google AI. (2025). Gemini API — Model Information.
<https://ai.google.dev/gemini-api/docs/models?hl=es-419>

Google AI. (2025). Gemini API — Rate Limits.
<https://ai.google.dev/gemini-api/docs/rate-limits?hl=es-419>

ANEXOS

El código fuente completo, los datos brutos y la documentación adicional están disponibles en el siguiente repositorio de GitHub:

[Repositorio de GitHub - Proyecto Final NLP](#)