

Índice:

Introducción …………………………………………………………………………………………………………………….. 2/6

Ejercicio 1 ….……………………………………………………………………………………………………………. 3, 4, 5 /6

# Introducción:

Este trabajo presenta la implementación de un chatbot experto en el juego de mesa 'The White Castle', como parte del Trabajo Práctico Final de la materia Procesamiento del Lenguaje Natural.

El objetivo principal fue desarrollar un sistema capaz de mantener conversaciones coherentes y informativas con un usuario sobre las reglas, estrategias y mecánicas del juego, basándose en una variedad de fuentes de conocimiento.

Este informe detalla las metodologías implementadas, el desarrollo del código, los resultados obtenidos y las conclusiones acerca del trabajo.

En este proyecto, se implementó un chatbot utilizando la técnica RAG, combinando modelos de lenguaje de gran tamaño con una base de conocimiento construida a partir de documentos de texto, datos tabulares y una base de datos de grafos. Se exploraron diferentes estrategias para la recuperación de información relevante, la clasificación de consultas y la generación de respuestas coherentes.

El presente informe detalla el proceso de desarrollo del chatbot, desde la recopilación y preparación de los datos hasta la evaluación de los resultados. Se presentan los desafíos encontrados y las soluciones implementadas, así como un análisis de los resultados obtenidos. Finalmente, se discuten las limitaciones del sistema y se proponen futuras líneas de trabajo.

# EJERCICIO 1:

## Resumen:

## Desarrollo (pasos para la resolución del problema, justificaciones):

### Instalación e importación de librerías

Se instalaron e importaron las librerías necesarias para la ejecución y realización del trabajo.

Además, se conecta con Drive para la descarga de los archivos que se van generando.

### Extracción de Información para las Bases de Datos

Además de la página propuesta por la cátedra (Board Game Geek) se realizó Web Scraping sobre las siguientes páginas web:

* The Opinionated Gamers: <https://opinionatedgamers.com/2023/09/25/dale-yu-review-of-the-white-castle/>
* Gaming Trend: <https://gamingtrend.com/feature/reviews/the-white-castle-review-a-tiny-box-with-a-ton-of-game/>
* Meeple Mpuntain: <https://www.meeplemountain.com/reviews/the-white-castle/>
* Punchboard: <https://www.punchboard.co.uk/the-white-castle-review/>

### Creación y acceso a las Bases de Datos

Para la Base de Datos Vectorial: se utilizó la información encontrada en las páginas detalladas anteriormente y en el foro de comentarios de Board Game Geek. Dicha información fue procesada (extraída, unificada, limpiada) para su correcto uso. Se crea un nuevo archivo que será de utilidad posteriormente. Se crea la base de datos vectorial en ChromaDB donde cada documento (fragmento de texto) está representado por un embedding numérico. Además, se almacenan metadatos como el ID y las palabras clave asociadas a cada documento. Esto permite realizar búsquedas semánticas eficientes basadas en la similitud entre los embeddings.

Para la Base de Datos tabular: se extrajo el número de jugadores y la edad mínima recomendada en Board Game Geek. Se crea una lista con esa información, que luego es transformada en un DataFrame para posteriormente ser descargado como archivo CSV (“contenido\_white\_castle.csv”).

Para la Base de Datos de Grafos: se extrae información de la parte de créditos en Board Game Geek, que contiene los diseñadores del juego, el artista, editoras, diseñadore gráfico, editores, categorías, mecanismos y familia. Cada uno de estos créditos serán un nodo hijo del nodo principal “The White Castle”, y cada uno de ellos contendrá sus respectivos datos. Se visualiza el grafo con Matplotlib.

### Modelo de embeddings

Se utiliza el modelo pre-entrenado all-MiniLM-L6-v2 de SentenceTransformers. Cada fragmento de texto generado por el RecursiveCharacterTextSplitter se pasa al modelo SentenceTransformer. El modelo procesa cada fragmento y produce un vector numérico de alta dimensión, que es el embedding correspondiente a ese fragmento.

Búsqueda semántica con Embeddings:

* Generar un Embedding para la Consulta: Cuando un usuario realiza una consulta, se genera un embedding para esa consulta utilizando el mismo modelo SentenceTransformer.
* Comparación de Embeddings: Se calcula la similitud entre el embedding de la consulta y los embeddings de todos los fragmentos de texto.
* Recuperación de Fragmentos Relevantes: Los fragmentos con los embeddings más similares a la consulta se consideran los más relevantes y se presentan al usuario como respuesta

### Consultas a las Bases de Datos

Las consultas a las Bases de Datos fueron realizadas mediante querys.

### Clasificador basado en LLM

Clase ZephyrChat: esta clase representa el chatbot en sí. Inicializa un historial de conversación vacío para almacenar las preguntas y respuestas del usuario y del asistente. La función generate\_response simula la generación de una respuesta basada en el historial completo.

Búsqueda BM25: la clase BM25Searcher implementa un algoritmo de búsqueda basado en BM25.Se utiliza para buscar documentos relevantes en un corpus basado en la similitud entre la consulta del usuario y los textos almacenados. La función retrieve recupera los documentos más relevantes para una consulta dada.

Construcción del Contexto: la función build\_context reúne información de distintas fuentes para enriquecer el contexto de la conversación:

* Resultados de la búsqueda BM25
* Resultados de la consulta a la base de datos ChromaDB (Reemplaza coleccion\_chromadb con tu instancia)
* Resultados de la consulta a un grafo de conocimiento (Reemplaza G con tu grafo cargado)
* Resultados de la consulta a datos tabulares (Reemplaza contenido\_white\_castle con tu dataframe)

Esta información se combina para proporcionar al modelo un contexto más completo para generar una respuesta adecuada.

Generación de la Respuesta: la función generate\_response\_with\_context toma la consulta del usuario y el contexto construido.

Manejo de la Consulta del Usuario: la función handle\_user\_query integra todas las partes anteriores: recibe la consulta del usuario, construye el contexto utilizando las diferentes fuentes de información y genera una respuesta basada en la consulta y el contexto.

## Conclusiones:

La implementación del chatbot aún no está completamente finalizada. Actualmente, se instancia el chatbot y, por otro lado, se realiza la búsqueda de información en diversas bases de datos, generando un contexto para cada respuesta. Sin embargo, lo ideal sería integrar un clasificador que permita seleccionar la base de datos más relevante para cada consulta, de manera que las respuestas se busquen solo en la base de datos correspondiente.

Aunque he generado varios modelos de referencia que podrían cumplir mejor con los objetivos del trabajo, opté por este enfoque para la entrega final debido a su estructura más clara y comprensible.

Algunas mejoras adicionales que podrían implementarse incluyen la integración del modelo de embeddings de la Unidad 3, de modo que el sistema seleccione una respuesta adecuada a partir de los resultados de la búsqueda y la devuelva.

Además, la implementación del Agente está sin resolver.

## Enlaces a modelos y librerías utilizadas:

Lista de herramientas empleadas junto con los enlaces a sus respectivas páginas oficiales:

* Selenium: <https://www.selenium.dev/documentation/>
* BeautifulSoup: <https://www.crummy.com/software/BeautifulSoup/bs4/css-selector-update-doc/>
* pandas: <https://pandas.pydata.org/docs/>
* docx: <https://python-docx.readthedocs.io/en/latest/user/documents.html>
* webdriver\_manager: <https://github.com/bonigarcia/webdrivermanager>
* chromadb: <https://pypi.org/project/chromadb/>
* rapidfuzz: <https://pypi.org/project/RapidFuzz/>
* langchain: <https://langchain-doc.readthedocs.io/en/latest/index.html>
* gdown: <https://pypi.org/project/gdown/>
* spaCy: <https://spacy.io/>
* SentenceTransformers: <https://huggingface.co/sentence-transformers>
* NLTK: <https://www.nltk.org/>