



Tecnicatura Universitaria en Inteligencia Artificial

Trabajo Práctico Final

Asignatura: Procesamiento del Lenguaje Natural

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

Alumno: Bravi Eugenio

Legajo: B-6600/1

Fecha de entrega: 29/06/2025

Índice

Resumen	3
Introducción.....	3
Metodología	4
Carga y Procesamiento de Datos	4
Text Splitting	4
Base de Datos Vectorial (ChromaDB)	6
BM25 Search	6
Reranker	7
Búsqueda Híbrida	7
Modelo de Lenguaje (LLM)	8
Base de Datos Tabular	8
Base de Datos de Grafos	9
Clasificador de Intención.....	10
Módulo de Recuperación (Retrieval)	11
Desarrollo e Implementación del Chatbot	11
Chatbot	11
Agente ReAct	11
Resultados y Evaluación	12
Pruebas del Chatbot	12
Pruebas del Agente ReAct	15
Conclusiones y Mejoras Futuras	17
Conclusiones	17
Mejoras Futuras.....	18
Referencias	19

Resumen

El presente informe detalla el desarrollo e implementación de un chatbot de preguntas y respuestas (QA) especializado en el juego de mesa "Parks". El sistema está diseñado para proporcionar información precisa y relevante a los usuarios, extrayendo datos de diversas fuentes estructuradas y no estructuradas. Se implementaron componentes clave como fragmentación de texto avanzada, una base de datos vectorial para búsqueda semántica, un motor de búsqueda BM25 para relevancia basada en palabras clave, un módulo de re-ranking para mejorar la pertinencia de los resultados, bases de datos tabular y de grafos para información específica, un clasificador de intención para enrutamiento inteligente de consultas, y un Agente ReAct que integra todas las herramientas y razona sobre el proceso de búsqueda. Se discuten las justificaciones técnicas para la elección de cada modelo y enfoque, se evalúa el rendimiento con casos de prueba y se proponen futuras mejoras.

Introducción

En la era digital, la capacidad de acceder rápidamente a información específica y precisa es fundamental. Los chatbots de preguntas y respuestas se han convertido en una herramienta invaluable para este propósito, especialmente en nichos de información especializados. Este proyecto aborda la creación de un sistema de QA para el popular juego de mesa "Parks", que implica manejar una variedad de tipos de datos, desde transcripciones de videos y foros hasta estadísticas estructuradas y relaciones complejas entre entidades del juego.

El desafío principal radica en la integración coherente de múltiples fuentes de información y en la capacidad de comprender la intención del usuario para dirigir la consulta a la base de conocimiento más adecuada. Este informe documenta el proceso de construcción de un chatbot robusto, utilizando técnicas avanzadas de Procesamiento del Lenguaje Natural (PLN) y arquitecturas de agentes inteligentes para ofrecer una experiencia de usuario fluida y precisa.

Metodología

La construcción del chatbot implicó la implementación y orquestación de diversos módulos, cada uno con una función específica en el pipeline de procesamiento de información.

Carga y Procesamiento de Datos

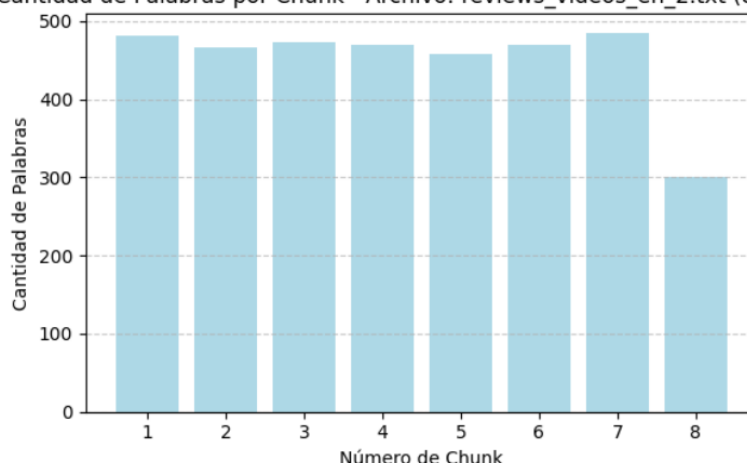
Los datos se obtuvieron de un archivo ZIP (“datos_parks.zip”) que contenía información textual en formato de documentos y transcripciones de videos, así como datos estructurados en formato CSV para estadísticas y relaciones. Los archivos de texto se cargaron y se realizó una limpieza inicial para eliminar duplicados basados en el contenido, asegurando la unicidad de la información procesada.

Text Splitting

La fragmentación del texto es crucial para optimizar la recuperación de información. Se utilizó un enfoque híbrido, adaptado a la naturaleza de cada tipo de documento:

1. Transcripciones de Videos (“videos”): Para estos documentos no estructurados, se empleó “TokenChunker” de la librería “chonkie”.
 - **Justificación:** Los transcritos de videos carecen de una estructura semántica clara (como párrafos o secciones bien definidos), lo que hace que los "semantic chunkers" sean menos efectivos. La fragmentación por tokens, con un tamaño fijo, permite dividir el texto en bloques manejables sin depender de la coherencia temática inherente del documento.
 - **Tamaño de Chunk:** Se estableció un tamaño de chunk de **512 tokens**. Este valor fue determinado experimentalmente: un tamaño mayor hacía que el buscador BM25 priorizara excesivamente estos archivos, mientras que uno menor resultaba en chunks demasiado cortos que perdían contexto crucial.

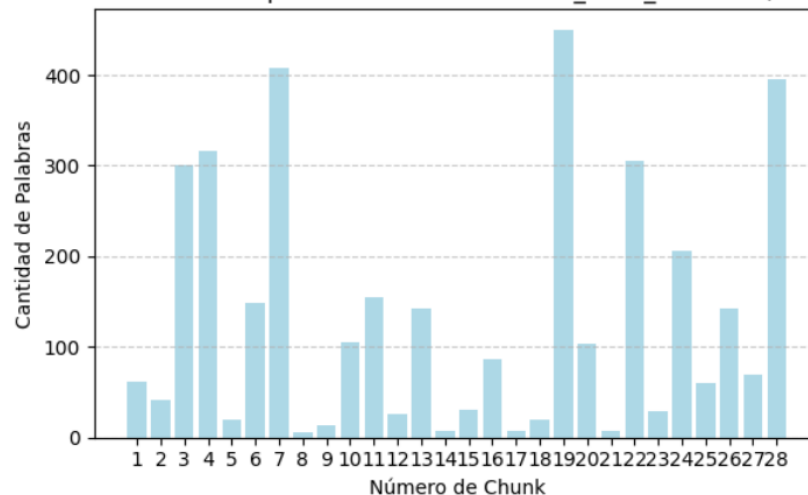
Cantidad de Palabras por Chunk - Archivo: reviews_videos_en_2.txt (8 chunks)



2. **Archivos de Texto Estructurados (reglas, reviews, guías):** Para estos documentos, se optó por “SemanticChunker” de “chonkie”.

- **Justificación:** Los textos estructurados (manuales de reglas, guías, foros) suelen tener una organización lógica que el “SemanticChunker” puede aprovechar. Este tipo de fragmentador agrupa oraciones con similitud semántica, preservando mejor el contexto temático dentro de cada chunk, a diferencia de los recursivos o por delimitadores fijos que pueden cortar ideas a la mitad.
- **Configuración del Semantic Chunker:** Se configuró con un “max_chunk_size” de **1024 tokens** y un “min_chunk_size” de **2 tokens**. Esto proporciona un margen amplio para que el modelo agrupe oraciones de manera inteligente, asegurando que los chunks no sean ni excesivamente cortos (perdiendo contexto) ni demasiado largos (diluyendo la especificidad).

Cantidad de Palabras por Chunk - Archivo: texto_foros_news.txt (28 chunks)



3. **Modelo de Embedding para Semantic Chunking:** Se utilizó “Qwen/Qwen3-Embedding-0.6B” de “HuggingFace”.

- **Justificación:** A diferencia del TP anterior que utilizaba [“intfloat/multilingual-e5-large-instruct”](#) (que tiene un límite de 512 tokens), [“Qwen/Qwen3-Embedding-0.6B”](#) fue seleccionado por su **mejor calidad de embeddings** según el [MTEB Leaderboard](#) y, crucialmente, por su capacidad de manejar textos con **más de 1024 tokens**. Esto permite generar chunks más grandes y contextualmente ricos, lo cual mejora la calidad de la recuperación semántica.

4. **Filtrado Post-Splitting:** Se observó que los chunks con menos de **500 caracteres** resultaban ser oraciones muy aisladas, con poca información relevante. Por ello, se

decidió descartar estos chunks para mejorar la calidad de la base de datos de documentos.

chunk_text	longitud
Many thanks!	12
\nYes indeed!	12
See here:\n\n	12
\nFrom page 8:\n	14
2. Yes\n3. Yep\n	14

Base de Datos Vectorial (ChromaDB)

Se empleó ChromaDB como base de datos vectorial para almacenar los embeddings de los chunks y permitir búsquedas de similitud semántica.

1. **Configuración:** La colección se configuró con “hnsw” y “space=“cosine””
 - **Justificación:** La similitud del coseno (“cosine”) es una métrica estándar y efectiva para comparar la orientación de vectores en espacios de alta dimensión, lo cual es ideal para los embeddings generados por modelos como Qwen
2. **Modelo de Embedding para Búsqueda:** El módulo “Vector_db” utiliza “SentenceTransformer(“Qwen/Qwen3-Embedding-0.6B”)” para generar los embeddings de las consultas.
 - **Justificación:** Es imperativo utilizar el **mismo modelo de embedding** tanto para la generación de los embeddings de los chunks como para las consultas. Esto asegura que tanto los datos almacenados como las consultas se representen en el mismo espacio vectorial, garantizando que las comparaciones de similitud sean consistentes y significativas.

BM25 Search

Se implementó un motor de búsqueda BM25 utilizando la librería “txtai.scoring.ScoringFactory”.

Justificación: BM25 es un algoritmo de relevancia basado en palabras clave que es altamente efectivo para capturar la “exactitud” léxica entre la consulta y los documentos. Es complementario a la búsqueda vectorial (semántica), ya que puede recuperar documentos que contienen términos clave exactos pero que podrían no ser semánticamente cercanos en

el espacio de embeddings, o viceversa. Esto proporciona una robustez crucial al sistema de recuperación.

Reranker

Para seleccionar los resultados más relevantes obtenidos por las búsquedas vectorial y BM25, se integró un modelo de re-ranking.

1. **Modelo de Reranker:** Se eligió “[BAAI/bge-reranker-v2-m3](#)” de la librería “FlagEmbedding”.
 - **Justificación:** Este modelo fue seleccionado por ser uno de los re-rankers más livianos disponibles que soporta el procesamiento de pares de texto (consulta-documento) con hasta **1024 tokens**. Esto es crucial dado que nuestros chunks máximos son de 1024 tokens, permitiendo que el re-ranker evalúe la relevancia contextual completa de los documentos sin truncamiento. Su eficiencia y calidad de re-ranking lo hacen ideal para el post-procesamiento de resultados de recuperación iniciales.
-

Búsqueda Híbrida

La búsqueda híbrida combina los resultados de la búsqueda vectorial y BM25, y luego aplica el re-ranker para obtener una clasificación final.

1. **Proceso:** Se obtienen los “max_results 2” documentos más relevantes de cada método (vectorial y BM25). Luego, se combinan, eliminando duplicados por “chunk_id” y marcando el “search_method” para cada documento por si es necesario debugear los documentos extraídos de cada base de datos. Finalmente, se envían todos los documentos combinados al re-ranker, que produce un nuevo score de relevancia. Los documentos se ordenan por este “rerank_score” y se devuelven los “max_results” principales.
2. **Justificación:** La combinación de búsqueda semántica y basada en palabras clave garantiza una cobertura más amplia de tipos de consultas y una mayor robustez frente a la variabilidad del lenguaje natural. El re-ranking final es vital porque los scores individuales de la búsqueda vectorial y BM25 no son directamente comparables entre sí. El re-ranker aprende a evaluar la relevancia conjunta,

proporcionando un ordenamiento más preciso de los documentos para la pregunta específica del usuario.

Modelo de Lenguaje (LLM)

Para las tareas de generación de texto y comprensión de lenguaje natural, se utilizó un LLM.

1. **Modelo:** “gemini-2.5-flash-preview-05-20” de Google Generative AI.
 - **Justificación:** Este modelo fue elegido por su **acceso gratuito**, lo cual es ideal para este proyecto. Además, ofrece un **rendimiento rápido** y la capacidad de generar **respuestas precisas** incluso para consultas complejas, sin requerir recursos computacionales locales. Su capacidad para manejar las distintas estructuras y complejidades de las consultas lo convierte en una opción sólida para las tareas de generación de código, clasificación de intención y respuesta final del chatbot.
-

Base de Datos Tabular

Para manejar datos estructurados (estadísticas del juego), se utilizó un DataFrame de Pandas.

1. **Procesamiento de Datos:** El CSV de estadísticas (“stats.csv”) fue cargado y transformado.

Título	Descripción
Avg. Rating	7.646
No. of Ratings	28,119
Std. Deviation	1.22

Se "pivoteó" el DataFrame para que las descripciones de las estadísticas se convirtieran en nombres de columna, facilitando el acceso directo a la información.

	avg_rating	no_of_ratings	std_deviation
Descripción	7.646	28119	1.22

2. **Mecanismo de Búsqueda:** El módulo “Tabular_db” utiliza el LLM para traducir consultas en lenguaje natural a código de Pandas.
3. **Justificación:** En lugar de construir un parser de consultas complejo, el LLM actúa como un “traductor”. Dada la estructura del DataFrame y la consulta del usuario, el LLM genera el fragmento de código de Pandas que, al ser ejecutado por la función “eval”, produce la respuesta deseada. Esto permite una flexibilidad enorme para consultar datos estadísticos sin la necesidad de reglas de consulta predefinidas y rígidas, aprovechando la capacidad de razonamiento del LLM sobre la estructura de datos.

Para gestionar las relaciones entre entidades del juego (diseñadores, publicadores, categorías), se implementó una base de datos de grafos.

- [illegible]

2. **Mecanismo de Búsqueda:** El módulo “Graph_db” también utiliza el LLM. Primero, el LLM identifica el tipo de relación (o relaciones) que el usuario está preguntando (ej.

"Designer", "Publisher"). Luego, se construye una consulta Cypher genérica que busca "PARKS" conectado a entidades a través de esa relación específica, y esta consulta se ejecuta en RedisGraph.

- **Justificación:** Los grafos son ideales para representar relaciones complejas. El LLM facilita la traducción de preguntas en lenguaje natural ("¿quién diseñó...", "¿a qué categoría pertenece...?") a las relaciones específicas en el grafo. Esta combinación permite responder preguntas sobre conexiones y atributos relacionales del juego de manera flexible y escalable.
-
-

Clasificador de Intención

Este componente es crucial para dirigir la consulta del usuario al módulo de búsqueda correcto.

1. **Implementación:** Se optó por una **LLM con Few-Shot Prompting** para la clasificación de intención.
 2. **Justificación:**
 - Se decidió armar el clasificador de intención utilizando una **LLM con Few-Shot Prompting**, a diferencia del **modelo entrenado en el punto 6 del TP1P2**. Esta elección se debe a que la LLM ofrece resultados correctos de manera consistente, un factor crítico para la fiabilidad del sistema. El modelo anterior mostraba una mayor inconsistencia, especialmente cuando la consulta no estaba directamente relacionada con el juego, pero compartía una estructura similar a las categorías definidas. Por ejemplo, ante una consulta como '¿Cuál es la cantidad de habitantes de París?', el modelo entrenado erróneamente respondía 'tabular_db'.
 - En contraste, la LLM logra clasificar correctamente este tipo de consultas, derivando a 'None' o la categoría adecuada. Dado que **este clasificador es una parte fundamental del pipeline de nuestro Chatbot, una decisión errónea aquí tendría un impacto significativo**: el retriever buscaría información en la base de datos incorrecta, resultando en respuestas erróneas o la ausencia total de respuesta. Por lo tanto, se priorizó la mayor precisión y robustez de la LLM para esta etapa crucial.
-
-

Módulo de Recuperación (Retrieval)

El módulo “Retrieval” actúa como el orquestador central del sistema de recuperación.

1. **Proceso:** Recibe la consulta del usuario, la envía al “ClasificadorIntencion”. Basándose en la intención clasificada (“graph_db”, “tabular_db”, “doc_db”, “None”), delega la consulta al módulo de búsqueda correspondiente (“Graph_db”, “Tabular_db”, “Hybrid_search”). Finalmente, formatea la respuesta obtenida de la base de datos seleccionada.
 2. **Justificación:** Centraliza la lógica de enrutamiento, haciendo que el sistema sea modular y fácil de mantener. Permite que cada base de datos se especialice en su tipo de información y que el sistema global decida dinámicamente cómo manejar cada consulta.
-
-

Desarrollo e Implementación del Chatbot

Chatbot

El chatbot principal (“ChatBotParks”) integra el LLM y el módulo de “Retrieval” para mantener una conversación.

Flujo:

1. El usuario ingresa una consulta.
 2. El chatbot invoca al “Retrieval” para obtener el contexto relevante.
 3. El contexto recuperado, junto con la consulta del usuario y el historial de conversación, se pasa al LLM (utilizando el “PROMPT_CHATBOT”).
 4. El LLM genera una respuesta basada en el contexto y el historial.
 5. El chatbot mantiene un historial de las interacciones (“self.history”) para permitir preguntas de seguimiento y mantener el contexto conversacional.
-
-

Agente ReAct

Para un manejo más sofisticado y multi-paso de las consultas, se implementó un **Agente ReAct** (“AgenteParks”).

1. **Patrón ReAct:** El agente sigue un ciclo “Thought -> Action -> Observation”.
 - **Thought:** El LLM razona sobre la consulta y decide qué acción tomar.
 - **Action:** El LLM especifica la herramienta a usar y sus parámetros (ej., “doc_search("Reglas importantes)").
 - **Observation:** El resultado de la ejecución de la herramienta se devuelve al LLM como nueva información.
 2. **Herramientas (“Tools”):** Una clase “Tools” agrupa todos los métodos de búsqueda: “graph_search”, “table_search”, “doc_search”, “wikipedia_search”, “duckduckgo_search”.
 - **Justificación de Herramientas Externas:** “wikipedia_search” y “duckduckgo_search” actúan como herramientas de fallback para información general que no esté presente en las bases de datos internas o para obtener datos actualizados (como fechas de lanzamiento). Esto amplía la cobertura de conocimiento del agente más allá de los datos proporcionados inicialmente.
 3. **“system_prompt”:** Un prompt detallado guía al LLM en el comportamiento del agente, enfatizando las reglas “Thought -> Action -> Observation”, el uso de una herramienta a la vez, el manejo de preguntas compuestas, el formato de las acciones, la no invención de respuestas y la verificación final.
 4. **Manejo de Iteraciones:** El agente puede realizar múltiples iteraciones “Thought -> Action -> Observation” hasta que considera que tiene suficiente información para dar una “Final Answer”.
 5. **Robustez:** Se incluyen validaciones de formato para las “Action” y manejo de errores en la ejecución de las herramientas, informando al LLM cuando una acción no es válida o una herramienta falla.
-
-

Resultados y Evaluación

Pruebas del Chatbot

El Chatbot demostró un buen desempeño para consultas directas y un solo paso, utilizando el clasificador de intención para enrutar a la base de datos correcta.

1. **Prueba 1: Consulta de Relación (graph_db)**
 - **Consulta:** “¿Qué relación tiene el juego PARKS con Bryce Cook?”
 - **Clasificación esperada:** “graph_db”
 - **Respuesta del Chatbot:** Según la información proporcionada, Bryce Cook es uno de los diseñadores de los insertos del juego PARKS.

- **Evaluación: Correcta.** El clasificador de intención derivó la consulta correctamente a la “graph_db”, y el LLM interpretó la relación recuperada para formular una respuesta clara y concisa.

2. Prueba 2: Consulta Tabular (tabular_db)

- **Consulta:** "¿El juego PARKS ha sido calificado por más de 10,000 usuarios en BoardGameGeek?"
- **Clasificación esperada:** “tabular_db”
- **Respuesta del Chatbot:** Sí, según la información proporcionada, el juego PARKS ha sido calificado por más de 10,000 usuarios en BoardGameGeek.
- **Evaluación:** Correcta. La pregunta fue clasificada adecuadamente para la “tabular_db”. El LLM generó el código Pandas necesario para realizar la verificación booleana y proporcionó una respuesta directa.

3. Prueba 3: Consulta Documental (doc_db)

- **Consulta:** "¿Qué diferencia hay entre las mecánicas del juego base PARKS y las de la expansion Nightfall?"
- **Clasificación esperada:** “doc_db”
- **Respuesta del Chatbot:** Según la información proporcionada, la expansión Nightfall introduce las siguientes mecánicas y cambios en el juego PARKS:
 - Nuevas cartas de Parque: Incluyen nuevas recompensas y arte inédito de la serie Fifty-Nine Parks Print.
 - Nuevas cartas de Año: Ofrecen nuevas oportunidades de puntuación de bonificación.
 - Nueva mecánica de Acampada (Camping): Permite experimentar el sendero de una forma completamente nueva.
- **Evaluación: Correcta.** La consulta fue enrutada a la “doc_db” y la búsqueda híbrida recuperó los chunks relevantes sobre la expansión Nightfall. El LLM sintetizó la información de los documentos para listar las diferencias de manera organizada.

4. Prueba 4: Consulta Fuera de Contexto (None)

- **Consulta:** "¿Cuál es la población actual de Islandia?"
- **Clasificación esperada:** “None”
- **Respuesta del Chatbot:** Este chatbot está especializado en responder preguntas sobre el juego de mesa Parks. Por favor, reformula tu pregunta para que esté relacionada con el juego.

- **Evaluación: Excelente.** El clasificador de intención con Few-Shot Prompting demostró su robustez al identificar correctamente que la consulta estaba fuera del ámbito del chatbot, evitando cualquier intento de búsqueda irrelevante y guiando al usuario.

5. Prueba 5: Consulta Compleja (graph_db)

- **Consulta:** "¿Quiénes son los publicadores de PARKS y en qué año se estrenó el juego?"
- **Clasificación esperada:** "graph_db"
- **Respuesta del Chatbot:** Según la información proporcionada:
 - El juego PARKS fue lanzado en el año 2019.
 - Los publicadores de PARKS son: Keymaster Games, Albi, Albi Polska, Feuerland Spiele, Lavka Games, Matagot, Surfin' Meeple China y Tranjis Games.
- **Evaluación: Correcta.** A pesar de contener dos piezas de información diferentes, ambas relacionadas con el grafo (publicadores y año de lanzamiento del juego base, que es un atributo en el grafo), el clasificador enrutó bien y el LLM fue capaz de extraer y presentar la información completa de la consulta al grafo.

6. Prueba 6: Consulta de Información Inexistente (doc_db)

- **Consulta:** "¿Me podrías explicar por qué cuando una persona tira los dados y sale 1 tres veces pierde un turno?"
 - **Clasificación esperada:** "doc_db" (pregunta sobre mecánicas)
 - **Respuesta del Chatbot:** Según el contexto proporcionado sobre el juego PARKS, no se menciona ninguna mecánica que involucre tirar dados, ni perder un turno por sacar un 1 tres veces. El juego se describe como uno de colección de sets y colocación de trabajadores que utiliza un tablero modular y recursos (tokens de agua, bosque, sol, montaña, fauna). Por lo tanto, no tengo información para explicar esa mecánica específica. Te sugiero que consultes las reglas del juego o reformules tu pregunta si te refieres a otro aspecto de PARKS.
 - **Evaluación: Excelente.** El chatbot clasificó correctamente la pregunta como documental. Al no encontrar información relevante en la "doc_db" sobre esa mecánica específica (porque no existe en Parks), el LLM generó una respuesta que honestamente indica la ausencia de la información y ofrece sugerencias al usuario, demostrando un comportamiento robusto ante la falta de datos.
-

Pruebas del Agente ReAct

El Agente ReAct sobresalió en la gestión de consultas complejas, que requieren múltiples pasos de razonamiento y el uso secuencial de varias herramientas.

- **Prueba 1: Consulta Compleja con Múltiples Herramientas y Fallback (graph_search, doc_search, duckduckgo_search)**
 - **Consulta:** "¿Quiénes son los publicadores del juego Parks y en qué año lanzo su última expansión?"
 - **Flujo simplificado:**
 - Thought 1: Identifica buscar publicadores (graph_search).
 - Action 1: `graph_search("publicadores de Parks")`
 - Observation 1: Lista los publicadores de Parks.
 - Thought 2: Identifica buscar el año de la última expansión (doc_search).
 - Action 2: `doc_search("última expansión Parks")`
 - Observation 2: Devuelve información sobre la expansión `Nightfall`.
 - Thought 3: Reconoce que la fecha de lanzamiento no está en los docs, intenta `duckduckgo_search`.
 - Action 3: `duckduckgo_search("Parks Nightfall expansion release date")`
 - Observation 3: Proporciona la fecha de lanzamiento de Nightfall.
 - Final Answer: Combina toda la información.
 - **Evaluación:** Muy buena. El agente demuestra la capacidad de desglosar la pregunta, usar `graph_search` para los publicadores, `doc_search` para encontrar el nombre de la expansión y, al no encontrar la fecha en la información interna, escalar a `duckduckgo_search` de manera autónoma y efectiva.
- **Prueba 2: Consulta en Idioma Diferente y Información Inexistente (doc_search)**
 - **Consulta:** "¿Could you explain to me why when a person rolls the dice and gets 1 three times, they lose a turn??"
 - **Flujo simplificado:**
 - Thought 1: Intenta buscar reglas relacionadas con los dados y perder turnos (doc_search).
 - Action 1: `doc_search("dice rolling lose turn Parks")`

- Observation 1: No se encuentran documentos relevantes sobre esa regla en el juego Parks.
 - Thought 2: Reconoce que la información no existe en el contexto del juego Parks.
 - Final Answer: Indica que esa regla no existe en Parks.
 - **Evaluación:** Excelente. El agente no inventa una respuesta y es capaz de reconocer cuando la información solicitada no aplica al juego, respondiendo de forma precisa y en el idioma de la consulta.
- **Prueba 3: Consulta Compleja con Múltiples Herramientas (doc_search, graph_search)**
- **Consulta:** "¿Cuales son las reglas mas importantes en Parks y a qué tipos de familia de juegos pertenece Parks?"
 - **Flujo simplificado:**
 - Thought 1: Identifica buscar reglas (doc_search).
 - Action 1: `doc_search("Reglas mas importantes en Parks")`
 - Observation 1: Proporciona chunks sobre reglas importantes.
 - Thought 2: Identifica buscar tipos de familia (graph_search).
 - Action 2: `graph_search("tipos de familia de juegos que pertenece Parks")`
 - Observation 2: Lista las categorías de familia de Parks.
 - Final Answer: Combina ambas informaciones.
 - **Evaluación:** Muy buena. El agente divide la pregunta en dos partes lógicas, utiliza las herramientas adecuadas para cada una y sintetiza una respuesta completa.
- **Prueba 4: Consulta Fuera de Contexto**
- **Consulta:** "¿Cual es la población actual de Islandia y cuantas personas juegan juegos de mesa?"
 - **Flujo simplificado:**
 - Thought 1: Reconoce que la pregunta no está relacionada con Parks.
 - Final Answer: Informa que el chatbot está especializado en Parks y no puede responder a esa pregunta.
 - **Evaluación:** Excelente. El agente sigue la regla de enfoque temático y no intenta buscar información irrelevante.
- **Prueba 5: Consulta Compleja Tabular (table_search multiple times)**

- **Consulta:** "¿Cual es el ranking del juego Parks? ¿cual es su rating promedio? y ¿Cuanto pesa?"
 - **Flujo simplificado:**
 - Thought 1: Busca el ranking (table_search).
 - Action 1: `table_search("ranking del juego Parks")`
 - Observation 1: Proporciona el ranking.
 - Thought 2: Busca el rating promedio (table_search).
 - Action 2: `table_search("rating promedio del juego Parks")`
 - Observation 2: Proporciona el rating promedio.
 - Thought 3: Busca el peso (table_search).
 - Action 3: `table_search("cuanto pesa el juego Parks")`
 - Observation 3: Proporciona el peso.
 - Final Answer: Combina las tres informaciones.
 - **Evaluación:** Excelente. El agente maneja múltiples consultas a la misma herramienta de manera secuencial y combina los resultados de forma coherente.
-
-

Conclusiones y Mejoras Futuras

Conclusiones

El proyecto ha logrado implementar un chatbot de preguntas y respuestas robusto y versátil para el juego "Parks". La arquitectura modular, que integra fragmentación de texto adaptativa, búsquedas híbridas (semántica y léxica), bases de datos especializadas (tabular, grafos) y un agente ReAct, ha demostrado ser muy efectiva.

Los puntos clave de éxito incluyen:

1. **Eficiencia en el Text Splitting:** La diferenciación entre `SemanticChunker` y `TokenChunker` y la justificación de los tamaños de chunk fueron cruciales para optimizar la granularidad de la información.
2. **Robustez del Reranker:** El uso de `BAAI/bge-reranker-v2-m3` permitió una mejora significativa en la relevancia de los resultados combinados de BM25 y la base vectorial.
3. **Precisión del Clasificador de Intención:** La decisión de usar un LLM con Few-Shot Prompting para la clasificación de intención fue fundamental para la fiabilidad del sistema, manejando eficazmente consultas dentro y fuera de contexto.

4. **Flexibilidad de las Bases de Datos Especializadas:** La integración del LLM para generar código Pandas y consultas Cypher permitió una interacción natural con datos estructurados y relaciones, sin necesidad de complejos parsers manuales.
 5. **Capacidad de Razonamiento del Agente ReAct:** El Agente ReAct demostró una excelente capacidad para desglosar preguntas complejas, seleccionar y utilizar herramientas de forma secuencial, y sintetizar respuestas coherentes, incluso escalando a herramientas externas cuando la información interna es insuficiente.
-
-

Mejoras Futuras

1. **Manejo de Errores Más Robusto en ``table_search`` y ``graph_search``:** Actualmente, si el LLM genera código Pandas o una relación Cypher incorrecta, la ``eval()`` o la consulta al grafo pueden fallar, resultando en un mensaje genérico.
 - a. **Mejora:** Implementar un mecanismo de "auto-corrección" o "retry with feedback". Si un ``eval()`` falla, el agente podría analizar el error, añadirlo al historial del prompt y pedir al LLM que genere un código corregido. Esto requeriría que el LLM sea capaz de entender errores de programación.
 - b. **Otra Mejora:** Proveer ejemplos de errores comunes de Pandas/Cypher al LLM en el prompt del agente para que pueda anticiparlos y corregirlos mejor.
 2. **Manejo de Ambigüedad y Preguntas de Seguimiento Complejas:** Aunque el chatbot mantiene historial, el agente ReAct podría beneficiarse de un manejo más explícito de la coreferencia y la ambigüedad en preguntas de seguimiento.
 - a. **Mejora:** Implementar un módulo de "historial enriquecido" que resuma el contexto relevante de las interacciones anteriores de manera más estructurada para el LLM, más allá de simplemente pasar el texto del diálogo. Esto podría implicar la extracción de entidades o la resolución de coreferencias en el turno actual con respecto a los turnos anteriores.
-
-

Referencias

- [MTEB Leaderboard](#): Hugging Face Spaces. (n.d.).
- [Qwen/Qwen3-Embedding-0.6B](#): Alibaba Cloud. (n.d.). Qwen/Qwen3-Embedding-0.6B. Hugging Face.
- [BAAI/bge-reranker-v2-m3](#): Beijing Academy of Artificial Intelligence. (n.d.). BAAI/bge-reranker-v2-m3. Hugging Face.
- [Chonkie](#): GitHub Repository. (n.d.). Chonkie: Semantic Chunker for LLMs.
- [txtai](#): GitHub Repository. (n.d.). txtai: Build AI-powered applications.
- [FlagEmbedding](#): GitHub Repository. (n.d.). FlagEmbedding: Reranker and Embedder from BAAI.
- [RedisGraph](#): Redis Labs. (n.d.). RedisGraph.
- [Google Generative AI](#): Google Developers. (n.d.). Gemini API.
- [ReAct \(Reasoning and Acting\)](#): Yao, S., et al. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. arXiv preprint arXiv:2210.03629.