

TRABAJO FINAL PROCESAMIENTO DE LENGUAJE NATURAL



Julián Gerónimo García

Febrero 2025

Tecnicatura Universitaria en Inteligencia Artificial

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Índice

| | | |
|-------|---|----|
| 1. | Introducción..... | 2 |
| 2. | Desarrollo Ejercicio 1..... | 3 |
| 2.1. | Setup General | 3 |
| 2.2. | Bases de Datos de Grafos | 3 |
| 2.3. | Bases de Datos Tabular | 6 |
| 2.4. | Extracción de Texto para Base Vectorial | 7 |
| 2.5. | Chunking | 8 |
| 2.6. | Generación de Embeddings + Base de Datos Vectorial..... | 9 |
| 2.7. | Clasificador | 9 |
| 2.8. | Retriever+ReRank | 10 |
| 2.9. | Query Dinámica Tabular | 11 |
| 2.10. | Query Dinámica para Grafos | 13 |
| 2.11. | Chatbot basado en LLM con UI | 15 |

1.Introducción

El objetivo principal de este trabajo es la implementación de un chatbot experto basado en la técnica Retrieval-Augmented Generation (RAG), aplicado al juego de mesa Azul, perteneciente a la categoría de Eurogames.

Para la construcción del chatbot, se integrarán diversas fuentes de conocimiento, incluyendo documentos de texto, datos tabulares y una base de datos de grafos. Además, se desarrollará un clasificador de consultas para determinar la fuente de información más relevante en cada interacción, evaluando dos enfoques distintos: uno basado en un modelo de lenguaje (LLM) y otro en un modelo entrenado con embeddings.

El trabajo también incorpora el concepto de agente inteligente, utilizando la arquitectura ReAct para la toma de decisiones en la recuperación de información.

A lo largo del informe, se describirá detalladamente el proceso de desarrollo, desde la extracción y preprocesamiento de datos hasta la implementación y evaluación del chatbot y su agente inteligente.

2.Desarrollo Ejercicio 1

Para el abordaje del armado del RAG se plantea el diagrama de flujo que se muestra a continuación, el cual se detallará en los siguientes puntos.

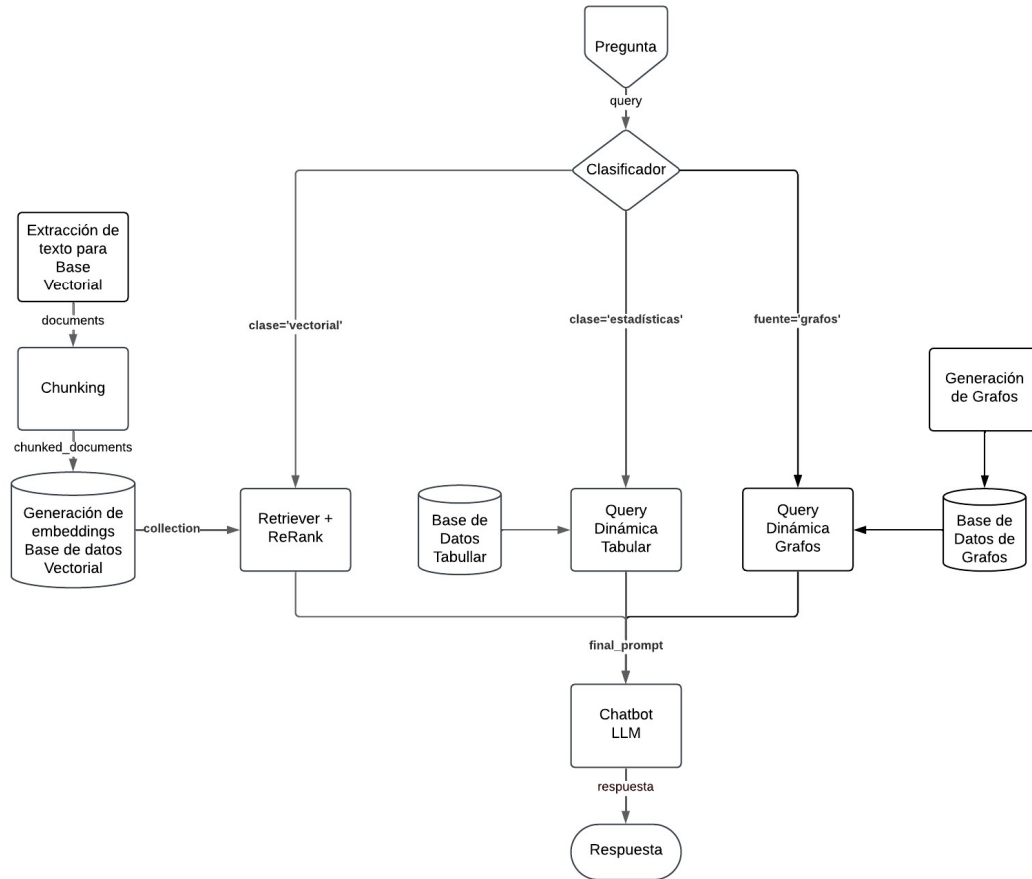


Figura 1 diagrama de flujo RAG

2.1. Setup General

En esta sección se clona el repositorio que contendrá las bases de datos que se generarán en el proyecto, se instalan las librerías y dependencias necesarias, y se definen funciones de uso global.

Entre las funciones de uso global se destaca **generate_answer**, que utiliza el modelo de LLM **zephyr-7b-beta** para la generación de respuestas a partir de un prompt determinado.

2.2. Bases de Datos de Grafos

El objetivo es crear un grafo por niveles, con el siguiente esquema:

- 2.2.1. Se crea un grafo con las relaciones obtenidas de la url <https://boardgamegeek.com/boardgame/230802/azul/credits>, en donde las tríadas que se formen tendrán como cabeza al nombre del juego, y se completarán con la información

encontrada. Por ejemplo, según la Figura 2, se formará la tríada [Azul]-[Designer]->[Michael Kiesling].

| | |
|---------------|--|
| Designer | Michael Kiesling |
| Solo Designer | N/A |
| Artists | Philippe Guérin Chris Quilliams |
| Publishers | Next Move Games Plan B Games Asmodee |

Figura 2 extracto de la url full credits

Para ello se utiliza la función **scrape_boardgame_credits(urls)** que retorna `df_credits`, y luego se crea el grafo con la función **create_graph**.

Se destaca la necesidad del uso de Selenium para acceder a los datos de la página web, ya que la misma se carga dinámicamente. Esto ocurrirá naturalmente para todas las extracciones que se hagan a urls del sitio boardgamegeek.com.



```
[160] 1 df_credits.head()
```

| | head | relation | tail | link |
|---|------|------------|------------------|---|
| 0 | Azul | Designer | Michael Kiesling | https://boardgamegeek.com/boardgamedesigner/42... |
| 1 | Azul | Artists | Philippe Guérin | https://boardgamegeek.com/boardgameartist/7407... |
| 2 | Azul | Artists | Chris Quilliams | https://boardgamegeek.com/boardgameartist/1405... |
| 3 | Azul | Publishers | Next Move Games | https://boardgamegeek.com/boardgamepublisher/3... |
| 4 | Azul | Publishers | Plan B Games | https://boardgamegeek.com/boardgamepublisher/3... |

Figura 3 df_credits

Notar que se almacena la url contenida en la cola, que se usará en el siguiente nivel para agregar más tríadas al grafo.

2.2.2. Con las urls mencionadas anteriormente, se scrapean los “Top Games” para cada url, y se agregan al grafo conectando las colas definidas en `df_credits` con los nombres de los juegos scrapeados, y considerando la misma relación que figura en `df_credits`.

Por ejemplo, para Michael Kiesling se generan 5 nuevas tríadas del siguiente tipo:

- [Azul: Summer Pavilion]<-[Designer]-[Michael Kiesling]
- ...
- [Vikings]<-[Designer]-[Michael Kiesling]

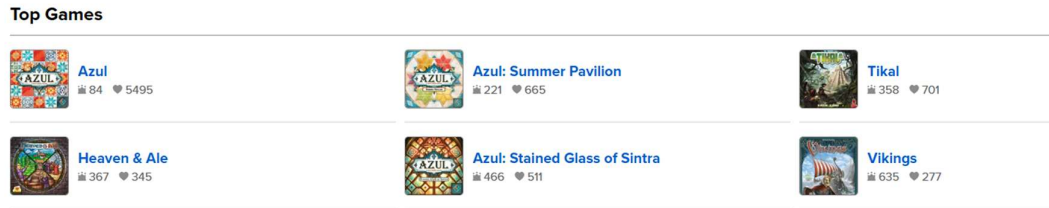


Figura 4 Top Games para Michael Kiesling

Notar que la tríada para el juego Azul ya existe, por lo tanto no se agrega.

Se utiliza la función **add_nodes**, la cual llama recursivamente a la función para scrapeo `crape_boardgame_top_game`.

2.2.3. El siguiente paso consiste en utilizar las mismas urls de `df_credits` pero para obtener las descripciones. Esta fuente de información resulta útil para obtener nuevas relaciones que alimenten el grafo de conocimiento.

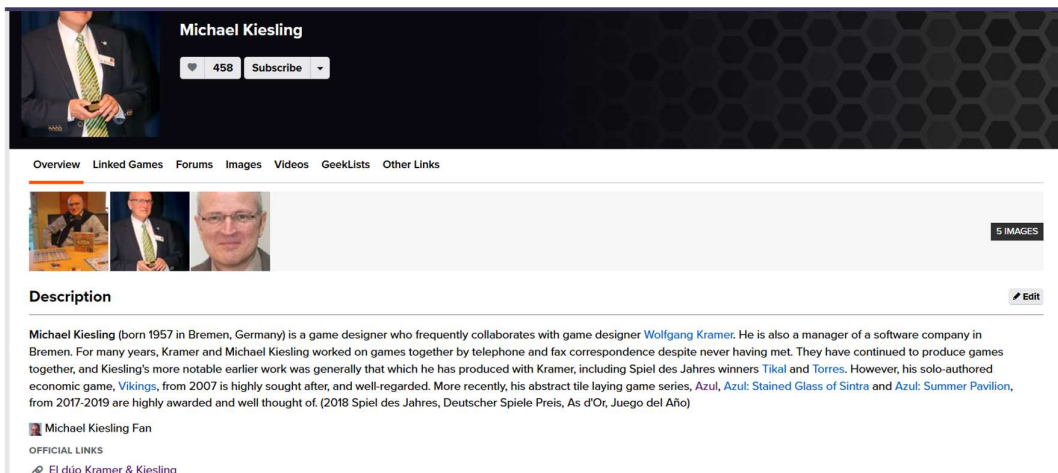


Figura 5 Description

El primer paso consiste entonces en scrapear la información, mediante la función **scrape_boardgame_description(url)**.

Luego, obtenida la descripción, se utiliza la función

prepare_prompt_for_triples_extraction(input_text: str, head: str) que recibe la descripción como `input_text`, la cabeza como `head` (Michael Kiesling continuando el ejemplo), y prepara el prompt para pasarlo a la función **generate_answer** para que genere las tríadas encontradas.

La función **process_boardgame_triples(df_credits, index)** anida las funciones anteriores y retorna un `df` con todas las tríadas halladas para una fila de `df_credits` en particular.

Se itera entonces para cada fila de `df_credits`, agregando las tríadas obtenidas en cada paso (que se encuentran en `df`) al grafo general, mediante la función **add_nodes_from_df(graph_name, df, head_col, relation_col, tail_col, host='localhost', port=6379)**.

Se obtiene así el grafo de la Figura 6, con un total de 250 triadas.
Se almacena la información en azul_graph.csv, para su posterior uso.

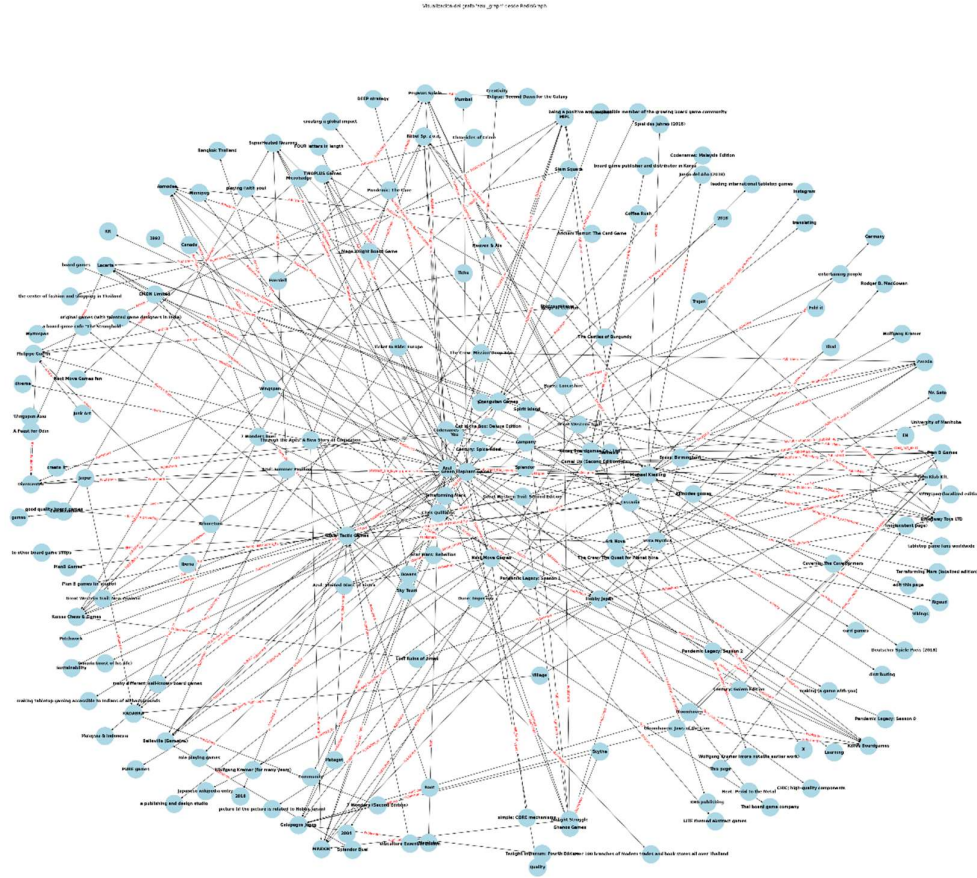


Figura 6 Grafo obtenido

2.3. Bases de Datos Tabular

Se genera el dataframe df_tabular a partir de la función **scrape_boardgame_tabular(urls)**, que recopila la información de la tabla ubicada en la url ['https://boardgamegeek.com/boardgame/230802/azul/stats'](https://boardgamegeek.com/boardgame/230802/azul/stats).

Se almacena la información en df_tabular.csv, para su posterior uso.



Figura 7 Stats

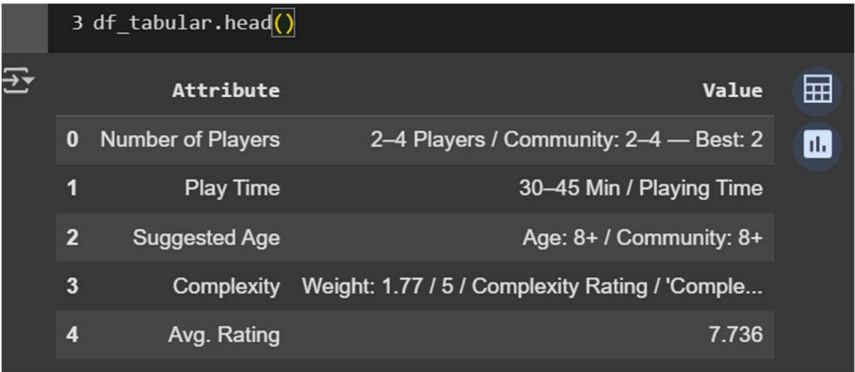


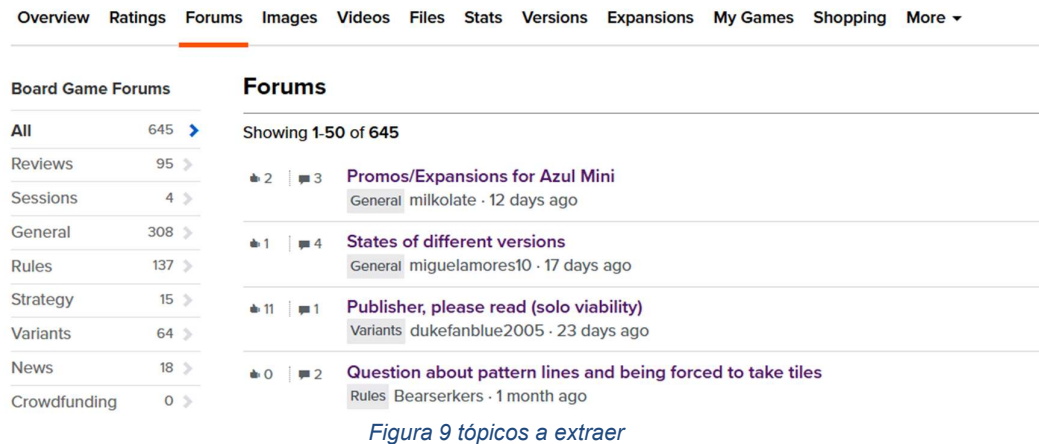
Figura 8 df_tabular

2.4. Extracción de Texto para Base Vectorial

Para el armado de la base vectorial se utilizan 4 fuentes de texto. Una vez obtenidos los 4 archivos de texto, se lo procesa con conversión a minúsculas y la aplicación de expresiones regulares para la eliminación de caracteres especiales, urls, menciones y hashtags.

2.4.1. Foro:

Se extraen los hilos de conversación del foro del sitio boardgamegeek (url <https://boardgamegeek.com/boardgame/230802/azul/forums/0>). Para ello se recopilan los 645 tópicos o temas mediante la función `scrape_all_topics(url)`.



La función **scrape_all_threads(topics)** extrae el hilo de conversación completo para cada tópico. Se almacena la información dentro de `general_threads.txt`. Solamente en este archivo se obtiene un total de 2.7M de caracteres, por lo que considerando un promedio de 2500 caracteres por página, suponen un total aproximado de 1000 páginas.

2.4.2. Reseñas:

Se scrapea el contenido del sitio <https://misutmeeple.com/2017/12/resena-azul/>, guardando el texto extraído en el archivo `resenas.txt`. A diferencia de las extracciones anteriores, este sitio no tiene carga dinámica por lo que se utiliza la librería BeautifulSoup para scrapear.

2.4.3. Comentarios:

De manera análoga al punto anterior, se obtiene contenido de comentarios del juego del sitio url = <https://misutmeeple.com/2017/12/resena-azul/>, y se guardan los datos en el archivo `comentarios.txt`.

2.4.4. Reglamento: se obtiene información del reglamento

`Azul_Quick_Rules_Guide.pdf` ubicado dentro de la documentación descargable del sitio de boardgamegeek. al cual se lo almacena en el archivo `reglamento.txt`. Se utiliza la librería PyPDF2 para acceder a la información.

2.5. Chunking

Una vez obtenidos y procesados los archivos de texto en el punto anterior, se utiliza `RecursiveCharacterTextSplitter` de `LangChain` para realizar la separación del texto en fragmentos o chunks de 500 caracteres, con un solape de 75 caracteres. Se investiga que esta configuración de valores resulta equilibrada para realizar búsquedas rápidas y precisas, pero con un largo adecuado para dar el contexto suficiente para no perder coherencia en los resultados.

2.6. Generación de Embeddings + Base de Datos Vectorial

Se generan los embeddings de los documentos citados en el punto anterior, utilizando el modelo **transformer distiluse-base-multilingual-cased-v2**.

Se utiliza ChromaDB para generar una base de datos vectorial persistente para posterior uso.

2.7. Clasificador

2.7.1. Clasificador basado en ejemplos y embeddings (clasificador_lr)

Se entrena un modelo de regresión logística multinomial para la clasificación de 300 preguntas según una de las tres categorías siguientes: vectorial [0], estadísticas [1] o grafos [2].

El conjunto de datos está balanceado (100 preguntas para cada categoría), y las mismas son vectorizadas mediante el mismo modelo mencionado en el punto 2.6 para generar el conjunto de entrenamiento.

Se obtienen las siguientes métricas, las cuales se consideran válidas. Se observa como oportunidad de mejora ampliar el conjunto de datos para mejorar aún más estos valores. Se genera la función de clasificación **clasificador_lr**.

| | | | | | |
|---|-----------|--------|----------|---------|--|
| Precisión Regresión Logística: 0.9 | | | | | |
| Reporte de clasificación Regresión Logística: | | | | | |
| | precision | recall | f1-score | support | |
| 0 | 0.85 | 0.85 | 0.85 | 20 | |
| 1 | 0.95 | 0.95 | 0.95 | 20 | |
| 2 | 0.90 | 0.90 | 0.90 | 20 | |
| accuracy | | | 0.90 | 60 | |
| macro avg | 0.90 | 0.90 | 0.90 | 60 | |
| weighted avg | 0.90 | 0.90 | 0.90 | 60 | |

Figura 10 métricas de clasificador_lr

2.7.2. Clasificador basado en LLM (clasificador_llm)

Se utiliza la función **prepare_classification_prompt(query_str: str)** para obtener la categoría correspondiente con la función **generate_answer**.

Se engloba el código dentro de la función **clasificador_llm**.

2.7.3. Elección de un clasificador

Se realizó la comparación de ambos clasificadores con un set de prueba de 20 preguntas nuevas.

Para el caso de clasificador_lr el acierto fue del 80% (16/20), mientras que para clasificador_llm fue del 45% (9/20).

Por este motivo, sumado a una mayor velocidad de procesamiento de clasificador_lr, se elige a clasificador_lr como modelo de clasificación.

No obstante la decisión tomada se entiende igualmente que es posible mejorar los desempeños de ambos clasificadores, aumentando las muestras para clasificador_lr, y afinando la preparación del prompt para clasificador_llm.

2.8. Retriever+ReRank

Se utiliza la función `vectorial_response(query, new_retriever, bm25_searcher, reranker)`, la cual realiza una búsqueda híbrida con reranking, y retorna la respuesta.

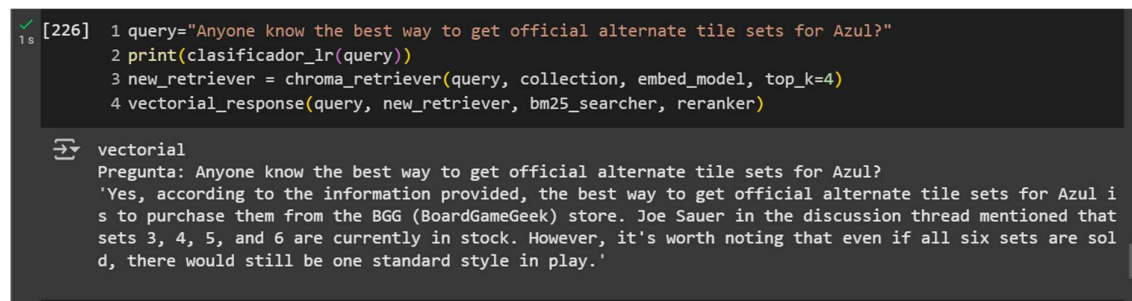
La búsqueda híbrida combina una búsqueda semántica y otra por palabras clave.

La búsqueda semántica se realiza en la base de datos vectorial generada con el método `retrieve` de `chroma`.

La búsqueda por palabras clave se hace por medio de la clase `BM25Searcher` creada a partir del modelo `BM25`, que tiene en cuenta la puntuación `TF-IDF` y el largo de los fragmentos de texto.

Finalmente se combinan ambas búsquedas y se las reordena por reranking para obtener los dos fragmentos más relevantes. Se utiliza el texto recuperado como contexto para la función `prepare_prompt(query, context_str)`, la cual genera el prompt que es consumido por la función `generate_answer(final_prompt)`, la cual retorna la respuesta que a su vez es retornada por `vectorial_response(query, new_retriever, bm25_searcher, reranker)`.

Ejemplos de uso:



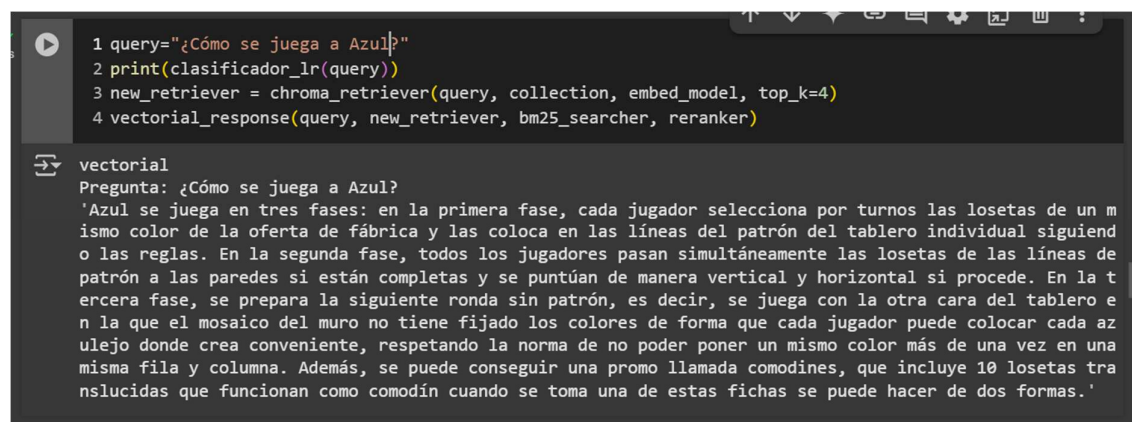
```
[226] 1 query="Anyone know the best way to get official alternate tile sets for Azul?"
      2 print(clasificador_lr(query))
      3 new_retriever = chroma_retriever(query, collection, embed_model, top_k=4)
      4 vectorial_response(query, new_retriever, bm25_searcher, reranker)
```

vectorial

Pregunta: Anyone know the best way to get official alternate tile sets for Azul?

'Yes, according to the information provided, the best way to get official alternate tile sets for Azul is to purchase them from the BGG (BoardGameGeek) store. Joe Sauer in the discussion thread mentioned that sets 3, 4, 5, and 6 are currently in stock. However, it's worth noting that even if all six sets are sold, there would still be one standard style in play.'

Figura 11 ejemplo de `vectorial_response`



```
1 query="¿Cómo se juega a Azul?"
2 print(clasificador_lr(query))
3 new_retriever = chroma_retriever(query, collection, embed_model, top_k=4)
4 vectorial_response(query, new_retriever, bm25_searcher, reranker)
```

vectorial

Pregunta: ¿Cómo se juega a Azul?

'Azul se juega en tres fases: en la primera fase, cada jugador selecciona por turnos las losetas de un mismo color de la oferta de fábrica y las coloca en las líneas del patrón del tablero individual siguiendo o las reglas. En la segunda fase, todos los jugadores pasan simultáneamente las losetas de las líneas de patrón a las paredes si están completas y se puntúan de manera vertical y horizontal si procede. En la tercera fase, se prepara la siguiente ronda sin patrón, es decir, se juega con la otra cara del tablero en la que el mosaico del muro no tiene fijado los colores de forma que cada jugador puede colocar cada azulejo donde crea conveniente, respetando la norma de no poder poner un mismo color más de una vez en una misma fila y columna. Además, se puede conseguir una promo llamada comodines, que incluye 10 losetas translúcidas que funcionan como comodín cuando se toma una de estas fichas se puede hacer de dos formas.'

Figura 12 ejemplo de `vectorial_response`

```
[224] 1 query="¿Azul permite varias estrategias?"
      2 print(clasificador_lr(query))
      3 new_retriever = chroma_retriever(query, collection, embed_model, top_k=4)
      4 vectorial_response(query, new_retriever, bm25_searcher, reranker)
```

vectorial

Pregunta: ¿Azul permite varias estrategias?

'La información de contexto indica que Azul permite algunas estrategias, pero afirma que el juego se gana o se pierde principalmente por las mejores opciones individuales y ofreciendo malas opciones a los oponentes. No se menciona una gran variedad de estrategias, por lo que se puede inferir que Azul permite algunas estrategias, pero no una gran variedad. Sin embargo, se menciona la posibilidad de notar que un color será escaso o abundante, lo que sugiere que es posible tener alguna estrategia relacionada con la disponibilidad de colores. En resumen, se puede responder que Azul permite algunas estrategias, pero la mayoría de las decisiones clave se basan en las mejores opciones individuales.'

Figura 13 ejemplo de `vectorial_response`

2.9. Query Dinámica Tabular

Se utiliza la función `tabular_response(query, df)` para obtener la respuesta a una query, a partir de la consulta a la base de datos tabular (df).

Dentro de dicha función, otra función llamada `prepare_prompt_query_dinamica(query_str: str, df)` genera el prompt necesario para que, a partir del texto de la pregunta, `generate_answer` devuelva como respuesta el filtro en pandas que es necesario aplicar a df. De dicho filtro se obtiene el valor numérico pretendido como respuesta.

Ejemplos de uso:

```
[234] 1 query="¿A cuántos jugadores les gusta el juego?"
      2 print(clasificador_lr(query))
      3 tabular_response(query, df_tabular)
```

estadísticas

'La respuesta es 5,481.'

Figura 14 ejemplo de `tabular_response`

```
[237] 1 query="¿Cuál es el rating promedio del juego?"
      2 print(clasificador_lr(query))
      3 tabular_response(query, df_tabular)
```

estadísticas

'La respuesta es 7.736.'

Figura 15 ejemplo de `tabular response`

```
[239] 1 query="¿Cuántos jugadores jugaron este juego?"
      2 print(clasificador_lr(query))
      3 tabular_response(query, df_tabular)
```

estadísticas

'La respuesta es 756,594.'

Figura 16 ejemplo de `tabular_response`

Tabla 1 *df_tabular*

| Attribute | Value |
|-------------------|---|
| Number of Players | 2–4 Players / Community: 2–4 – Best: 2 |
| Play Time | 30–45 Min / Playing Time |
| Suggested Age | Age: 8+ / Community: 8+ |
| Complexity | Weight: 1.77 / 5 / Complexity Rating / 'Comple... |
| Avg. Rating | 7.736 |
| No. of Ratings | 96,923 |
| Std. Deviation | 1.16 |
| Weight | 1.77 / 5 |
| Comments | 12,119 |
| Fans | 5,481 |
| Page Views | 3,673,530 |
| Overall Rank | 83 |
| Abstract Rank | 2 |
| Family Rank | 13 |
| All Time Plays | 756,594 |
| This Month | 2,063 |
| Own | 160,202 |
| Prev. Owned | 6,750 |
| For Trade | 894 |
| Want In Trade | 966 |
| Wishlist | 14,818 |
| Has Parts | 39 |
| Want Parts | 27 |

Relaciones encontradas en el grafo:

(Azul)-[Artists]->(Chris Quilliams)
 (Chris Quilliams)-[from]->(Winnipeg)
 (Chris Quilliams)-[from]->(Manitoba)
 (Chris Quilliams)-[from]->(Canada)
 (Chris Quilliams)-[lived]->(Canada (most of his life))
 (Chris Quilliams)-[studied]->(University of Manitoba)
 (Chris Quilliams)-[lives]->(Rigaud)
 (Chris Quilliams)-[works_for]->(Plan B games (in studio))
 (Pandemic Legacy: Season 1)-[Artists]->(Chris Quilliams)
 (Great Western Trail: Second Edition)-[Artists]->(Chris Quilliams)
 (Pandemic Legacy: Season 2)-[Artists]->(Chris Quilliams)
 (Great Western Trail: New Zealand)-[Artists]->(Chris Quilliams)
 (Iberia)-[Artists]->(Chris Quilliams)

Obtenido el subgrafo de relevancia se lo convierte a string para pasarlo a la función **prepare_prompt(query, context_str)** como contexto, que a su vez alimenta a la función **generate_answer(final_prompt)** que devuelve la respuesta retornada por **graph_response**.

Ejemplos de uso:

```
[248] 1 query = "¿en que juegos participo Chris Quilliams?"
      2 print(clasificador_lr(query))
      3 graph_response(query, graph)
```

grafos
 'Chris Quilliams participa en los juegos de arte de diseño de los siguientes títulos: Pandemic Legacy: Season 1, Pandemic Legacy: Season 2, Great Western Trail: Segunda Edición, Great Western Trail: Nueva Zelanda y Iberia.'

Figura 19 ejemplo de graph_response

```
[249] 1 query = "¿Dónde nació Michael Kiesling?"
      2 print(clasificador_lr(query))
      3 graph_response(query, graph)
```

grafos
 'La información indica que Michael Kiesling nació en Alemania.'

Figura 20 ejemplo de graph_response

```
[253] 1 query = "¿Qué juegos publicó Plan B Games?"
      2 print(clasificador_lr(query))
      3 graph_response(query, graph)
```

grafos
 'Plan B Games publicó los siguientes juegos según la información de contexto proporcionada: Azul, Next Move Games (perteneciente a Plan B Games), Great Western Trail, Village, Camel Up (Segunda Edición), Century: Golem Edition y Century: Spice Road.'

Figura 21 ejemplos de graph_response

2.11. Chatbot basado en LLM con UI

Se implementó un chatbot inteligente utilizando Gradio como interfaz para la interacción con los usuarios. La arquitectura del chatbot está diseñada para clasificar las consultas en las tres categorías vistas, y responder en consecuencia utilizando las funciones vistas en los puntos 2.8, 2.9, 2.10, mediante el uso de la función **chat_function(message, history)**.

Se realizan distintas preguntas que se muestran a continuación, con un tiempo promedio de respuesta de 6 segundos, lo cual se considera aceptable dado que aún no se realizó ninguna optimización.

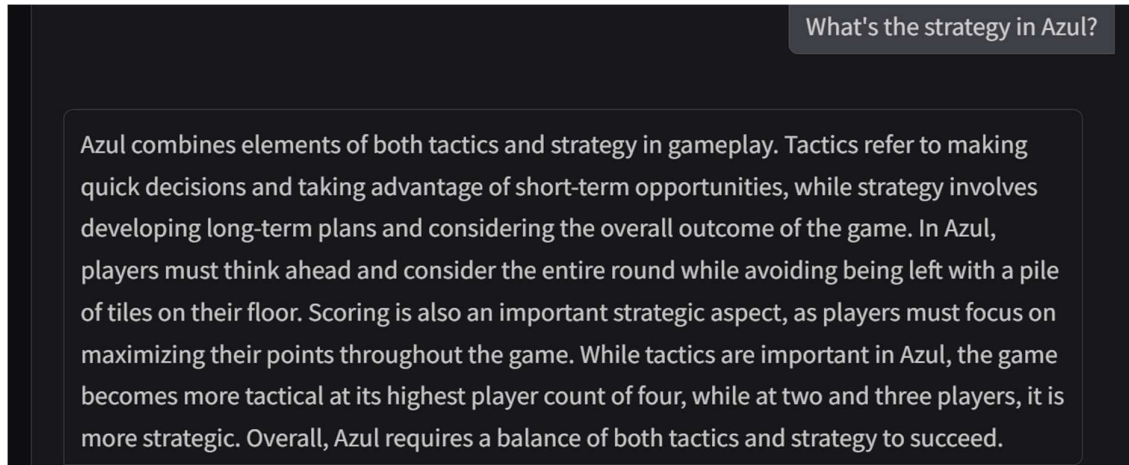


Figura 22 chatbot

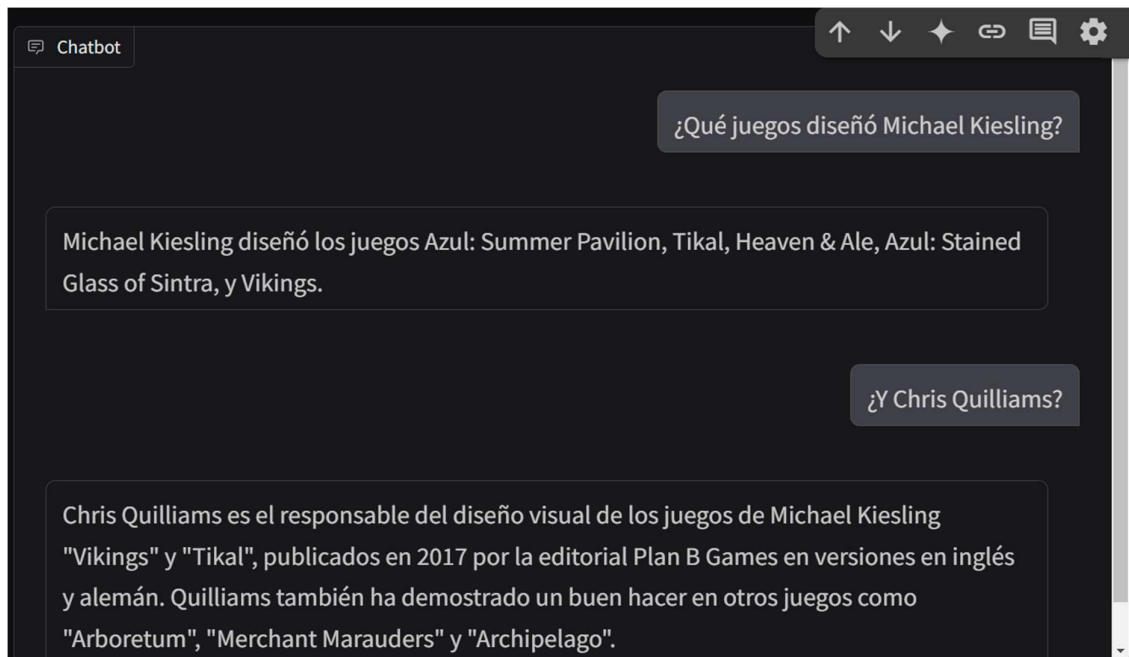


Figura 23 chatbot



Figura 24 chatbot

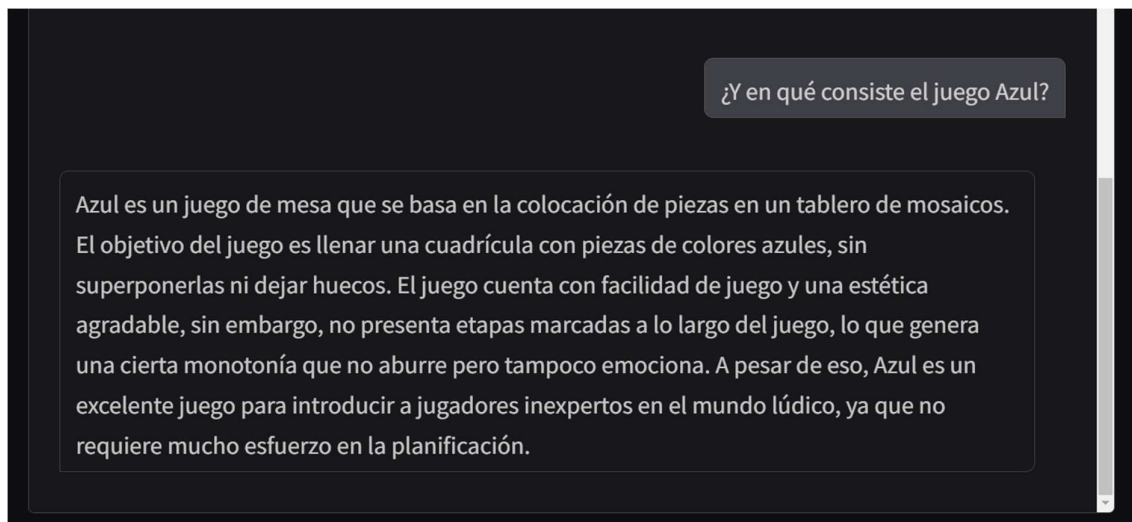


Figura 25 chatbot

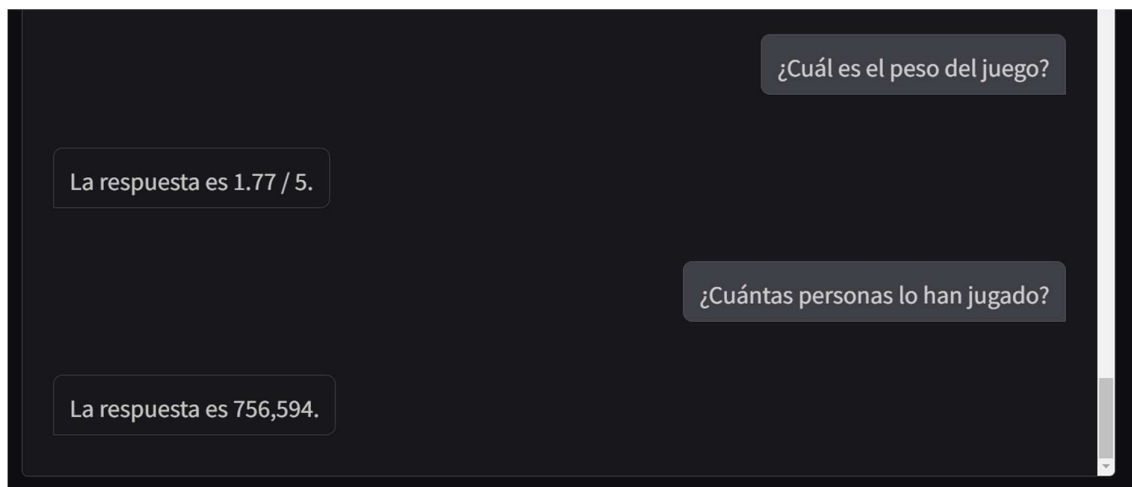


Figura 26 chatbot

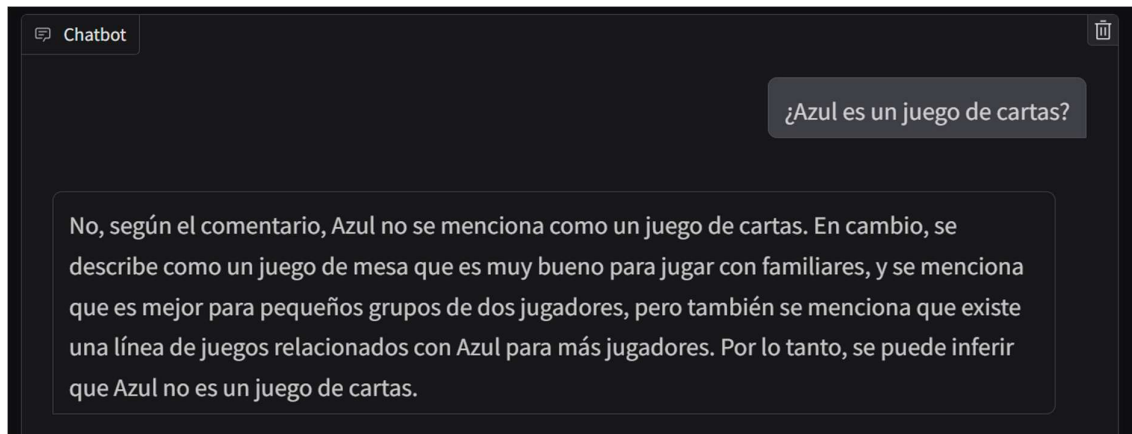


Figura 27 chatbot