

# librerIA Chatbot

SISTEMAS DE INFORMACIÓN DE GESTIÓN Y  
BUSSINESS INTELLIGENCE, 2024-2025

ÁLVARO PRIETO ÁLVAREZ

# Índice de contenidos

<b>ÍNDICE DE ILUSTRACIONES .....</b>	<b>2</b>
<b>GLOSARIO DE TÉRMINOS .....</b>	<b>3</b>
<b>1 INTRODUCCIÓN .....</b>	<b>7</b>
1.1 OBJETIVO Y DESCRIPCIÓN DEL PROBLEMA .....	7
<b>2 HERRAMIENTAS UTILIZADAS .....</b>	<b>8</b>
<b>3 DESARROLLO .....</b>	<b>10</b>
3.1 APRENDIZAJE Y FAMILIARIZACIÓN .....	10
3.2 BÚSQUEDA DEL DATASET .....	11
3.3 CREACIÓN DE UNA BASE DE DATOS .....	12
3.4 GENERACIÓN DE LOS EMBEDDINGS .....	13
3.5 CREACIÓN DE AGENTES REACT CON HERRAMIENTAS RAG .....	16
3.6 DESARROLLO DE LA INTERFAZ WEB .....	19
3.7 INTEGRACIÓN DE DICTADO POR VOZ. WHISPER .....	21
3.8 REALIZACIÓN DE PRUEBAS DE FUNCIONAMIENTO .....	22
<b>4 RESULTADOS Y CONCLUSIONES .....</b>	<b>24</b>
4.1 ANÁLISIS DE RESULTADOS .....	24
4.1.1 Recomendación de libros genérica con LLM grande .....	24
4.1.2 Recomendación de libros genérica con LLM mediano .....	25
4.1.3 Resolución de dudas acerca de un libro dado .....	26
4.1.4 Recomendación de libros similares a uno dado .....	28
4.1.5 Recomendación de libros en inglés .....	28
4.2 ANÁLISIS D.A.F.O. ....	29
4.2.1 Debilidades .....	29
4.2.2 Amenazas .....	30
4.2.3 Fortalezas.....	30
4.2.4 Oportunidades.....	31
4.3 LÍNEAS DE FUTURO .....	31
4.4 CONCLUSIONES.....	32
<b>5 BIBLIOGRAFÍA.....</b>	<b>35</b>

## Índice de ilustraciones

FIGURA 3.6.1 - INTERFAZ DE LA APLICACIÓN. CHAT DE STREAMLIT. ELABORACIÓN PROPIA .....	20
FIGURA 3.7.1 - INTERFAZ DE LA APLICACIÓN CON BOTÓN PARA TRANSCRIBIR. ELABORACIÓN PROPIA .....	22
FIGURA 4.1.1 – RECOMENDACIÓN DE LIBROS SIMILARES A UNO DADO. ELABORACIÓN PROPIA.....	24
FIGURA 4.1.2 - LOGS DEL AGENTE RECOMENDANDO LIBROS SIMILARES A UNO DADO. ELABORACIÓN PROPIA .....	25
FIGURA 4.1.3 – RECOMENDACIÓN DE LIBROS SIMILARES A UNO DADO. ELABORACIÓN PROPIA.....	25
FIGURA 4.1.4 - LOGS DEL AGENTE AL RECOMENDAR UN LIBRO Y QUEDARSE EN BUCLE. ELABORACIÓN PROPIA.....	26
FIGURA 4.1.5 - DATOS EN LA BBDD ORIENTADA A GRAFOS Y SUS RELACIONES. ELABORACIÓN PROPIA .....	26
FIGURA 4.1.6 - CONVERSACIÓN PARA OBTENER INFORMACIÓN DE DOS EJEMPLARES. ELABORACIÓN PROPIA.....	27
FIGURA 4.1.7 - RECOMENDACIÓN DE LIBROS A PARTIR DE UNO DADO. ELABORACIÓN PROPIA .....	28
FIGURA 4.1.8 - RECOMENDACIÓN DE LIBROS USANDO INGLÉS. ELABORACIÓN PROPIA.....	28
FIGURA 4.1.9 - LOGS AL SOLICITAR RECOMENDACIÓN DE LIBRO EN INGLÉS. ELABORACIÓN PROPIA .....	29

## Glosario de términos

<b>API</b>	<i>Application Programming Interface</i> . Interfaz de programación de aplicaciones, es un software intermediario que permite que las aplicaciones se comuniquen entre sí [1].
<b>BBDD</b>	Base de Datos.
<b>Chatbot</b>	Programa informático que simula la conversación humana con el usuario final. Suelen utilizar técnicas de IA conversacional como el procesamiento el lenguaje natural. [2]
<b>Cypher</b>	Lenguaje de Neo4j declarativo de consultas, que está especializado en permitir las acciones propias de una base de datos de grafos de Neo4j.
<b>CSV</b>	<i>Comma Separated Values</i> . Valores Separados por Comas. Tipo de fichero tabular, que puede tener encabezados, cuyas celdas se distinguen por un separador, típicamente una coma (",") u otro carácter no alfanumérico. Entre dos filas consecutivas existe un salto de línea.
<b>Dataset</b>	Conjunto de datos, generalmente usado para ciencia de datos y/o algoritmos de inteligencia artificial.
<b>Frontend</b>	También visto como Front End, es el lado del cliente en una aplicación web, la parte de esta que interactúa con los usuarios.

<b>GitHub</b>	Plataforma online de desarrollo colaborativo para alojar proyectos software utilizando el sistema de control de versiones <i>Git</i> . [3]
<b>GPU</b>	<i>Graphics Processing Unit</i> . Unidad de Procesamiento Gráfico. Procesador dedicado a trabajar con gráficos y operaciones en coma flotante para aligerar la carga de trabajo de la unidad central de procesamiento (CPU).
<b>Hugging Face</b>	Hugging Face Inc, empresa estadounidense que desarrolla herramientas para crear aplicaciones de IA y aprendizaje profundo. Entre otras, ofrece bibliotecas en Python para integrar modelos varios y poder utilizarlos. [4]
<b>IA</b>	Inteligencia Artificial. Conjunto de algoritmos y funciones que permiten a los ordenadores realizar tareas que pueden parecer propias de los seres humanos. Algunos ejemplos son el reconocimiento óptico de caracteres o planificación de itinerarios para viajes.
<b>Jupyter Notebook</b>	Plataforma informática interactiva basada en web que combina código en directo, ecuaciones, texto narrativo, visualizaciones, etc. Soporta más de 40 lenguajes de programación como Python o R. [5]
<b>Kaggle</b>	Empresa subsidiaria de Google LLC, es una comunidad en línea de científicos de datos y profesionales del aprendizaje automático [6]. Ofrece, entre otros servicios, <i>datasets</i> gratuitos para que los trabajadores de ciencia de datos puedan utilizar para desarrollar diferentes proyectos.
<b>LangChain</b>	Framework para la integración de LLMs.

<b>LlamaIndex</b>	Framework para construir aplicaciones de IA generativa aumentada por contexto con LLMs, incluyendo agentes y flujos de trabajo. [7]
<b>LLM</b>	<i>Large Language Model</i> . Modelo Extenso de Lenguaje. Modelos de aprendizaje profundo basados en texto muy grandes que se <i>preentrenan</i> con ingentes cantidades de datos. Tiene capacidad de autoatención y comprender relaciones entre palabras y frases. [8]
<b>Neo4j</b>	Sistema gestor de base de datos orientadas a grafos.
<b>Ollama</b>	Es una herramienta de código abierto que permite ejecutar localmente grandes modelos de redes neuronales, como LLMs. [9]
<b>Python</b>	Lenguaje de programación de alto nivel multiparadigma interpretado que se enfoca en la legibilidad del código. [10]
<b>RAG</b>	<i>Retrieval Augmented Generation</i> . Generación Aumentada por Recuperación. Proceso de optimización de la salida de un modelo lingüístico de gran tamaño, de modo que haga referencia a una base de conocimientos autorizada fuera de los orígenes de datos de entrenamiento antes de generar una respuesta. [11]
<b>Recomendador</b>	Sistema informático automático que aconseja acerca de un tema concreto. Por ejemplo, Spotify permite reproducir música en función de los gustos de cada usuario de forma automática, creando listas personalizadas individuales [12].

<b>Script</b>	Código o secuencia de comandos usado para automatizar tareas. Generalmente son cortos y forman parte de una aplicación mayor.
<b>Streamlit</b>	Framework en Python de código abierto para que científicos de datos, IA y aprendizaje automático desarrollen aplicaciones de datos interactivas en sólo unas pocas líneas de código. [13]
<b>TensorFlow</b>	Interfaz para expresar algoritmos de aprendizaje automático y una implementación para ejecutar dichos algoritmos. Un cómputo expresado con TensorFlow se puede ejecutar con poco o ningún cambio en una amplia variedad de sistemas heterogéneos, desde dispositivos móviles como teléfonos y tabletas hasta sistemas distribuidos a gran escala de cientos de máquinas y miles de dispositivos computacionales como tarjetas GPU. [14]
<b>TPU</b>	<i>Tensor Processing Unit</i> . Unidad de Procesamiento Tensorial. Chip desarrollado por Google optimizado para el aprendizaje automático con redes neuronales artificiales mediante el uso de TensorFlow. [15]
<b>Word embedding</b>	Conjunto de técnicas de aprendizaje y modelos de lenguaje en procesamiento del lenguaje natural (rama de IA) donde palabras o frases del lenguaje natural – humano – son representadas como vectores de números reales— <i>embeddings</i> . [16]

# 1 Introducción

En un mundo conectado, la cantidad de información y opciones disponibles supera con creces la capacidad humana de procesarlas eficientemente. Desde el comercio electrónico hasta el consumo de contenido digital, los usuarios enfrentan diariamente una sobrecarga cognitiva al intentar encontrar productos, servicios o información relevantes en un océano de posibilidades. En este contexto, los sistemas *recomendadores* han emergido como herramientas fundamentales para simplificar la toma de decisiones, personalizar experiencias y aumentar la satisfacción del usuario.

Estos sistemas, impulsados por avances en inteligencia artificial y técnicas de RAG no solo reducen la complejidad en la búsqueda de información, sino que también permiten a las empresas mejorar la retención de clientes, optimizar sus estrategias de marketing y potenciar sus ingresos. Desde la sugerencia de películas en plataformas de *streaming* hasta la personalización de carritos de compras en tiendas virtuales, los sistemas *recomendadores* no solo facilitan la vida cotidiana, sino que además generan valor en términos de eficiencia y satisfacción.

Este es el trabajo final de la asignatura de Sistemas de Información de Gestión y Business Intelligence del cuarto curso del Grado en Ingeniería Informática de la Universidad de León, que toma como base lo anteriormente expuesto y tiene como objetivo desarrollar un sistema *recomendador*.

## 1.1 Objetivo y descripción del problema

Al principio de la asignatura, se nos planteó a los alumnos el reto de desarrollar un sistema *recomendador*. La pregunta en sí misma, como se puede apreciar, es excelsamente abierta. En cambio, se nos recomendaron herramientas que podían ser de nuestro interés como Neo4j, LlamaIndex, Chat-GPT o Kaggle.

Dado que la temática es completamente libre, opté por elegir un tema sencillo para que este trabajo fuera simplemente una prueba de concepto, en lugar de un producto final. Es por ello por lo que muy probablemente haya multitud de diversos puntos tanto en el propio código como en los conceptos utilizados que sean mejorables o, directamente, se deban enfocar desde otra perspectiva.

En cambio, viendo que la mayoría de aplicaciones similares utilizan APIs externas y necesitan conexión a Internet, me propuse desarrollar todo en local. De este modo, no sería necesario tener conectividad. A la vez que se mantiene la gobernanza de los datos.

He aquí pues el resultado del trabajo que he venido realizando a lo largo de la asignatura, y que pasaré a exponer más detalladamente en los siguientes puntos de esta memoria.



## 2 Herramientas utilizadas

Aunque se desarrollarán más detalladamente en el punto 3, y algunas de ellas ya han sido mencionadas en el Glosario de términos, aquí se hace una breve enumeración de las distintas tecnologías específicas utilizadas durante este proyecto.

- **GitHub:** plataforma para control de versiones y colaboración en el desarrollo de software, utilizada para almacenar y gestionar código de manera eficiente.
- **LlamaIndex:** *framework* para integrar datos externos en modelos de lenguaje, optimizando la interacción con bases de datos y documentos no estructurados.
- **Ollama:** plataforma especializada en el despliegue de modelos de lenguaje personalizados para casos de uso específicos.
- **Agentes ReACT:** metodología que combina razonamiento y acción para construir agentes autónomos que toman decisiones basadas en modelos de lenguaje.
- **Neo4j:** base de datos orientada a grafos que permite modelar y consultar relaciones complejas entre datos de forma eficiente.
- **Kaggle:** comunidad en línea para el aprendizaje de ciencia de datos y aprendizaje automático, conocida por sus competencias y conjuntos de datos públicos.
- **Streamlit:** *framework* de Python para crear aplicaciones web interactivas enfocadas en visualización de datos y prototipos rápidos.
- **Whisper:** modelo de transcripción automática desarrollado por OpenAI, diseñado para convertir audio en texto con alta precisión.
- **LLMs:** modelos avanzados de procesamiento de lenguaje natural, entrenados con grandes volúmenes de datos para realizar tareas complejas.
- **Python:** lenguaje de programación versátil y ampliamente utilizado, especialmente en ciencia de datos, inteligencia artificial y desarrollo web.
- **Hugging Face:** plataforma y comunidad para desarrollar, compartir y utilizar modelos de aprendizaje automático, con un enfoque en procesamiento de lenguaje natural.
- **Jupyter Notebook:** entorno interactivo que permite combinar código, texto explicativo y visualizaciones en un único documento.
- **LangChain:** *framework* para construir aplicaciones basadas en modelos de lenguaje, integrando memoria y flujos de trabajo complejos.
- **Google Colab:** servicio basado en la nube para ejecutar código Python en “cuadernos”, con acceso a recursos de computación como GPUs.

- **Cypher:** lenguaje de consulta específico para bases de datos de grafos, diseñado para interactuar con sistemas como Neo4j.
- **Visual Studio Code:** editor de código ligero y altamente extensible, utilizado para el desarrollo de aplicaciones en múltiples lenguajes de programación.
- **Graph Data Science (GDS):** biblioteca para Neo4j que contiene una gran cantidad de algoritmos sobre grafos, divididos por distintas categorías y optimizados para su uso en la BBDD.

## 3 Desarrollo

### 3.1 Aprendizaje y familiarización

El desarrollo del proyecto comenzó con la propuesta por parte del profesor de la asignatura. El **primer paso consistía en aprender**. Familiarizarse con diversas tecnologías completamente nuevas para los alumnos que no habíamos visto—ni tan si quiera oído hablar—durante los cursos del grado. Como ya se expuso anteriormente, algunas de las tecnologías las mencionó el profesor.

Comenzando por las bases de datos vectoriales, de las que solo conocía su existencia, se nos recomendó que aprendiéramos **Neo4j**. Aunque, como se estableció desde el principio, el proyecto era completamente abierto, luego no era necesario usarla. Pero podía constituir una buena herramienta gestora de base de datos con la cual comenzar. En la sección de conclusiones, se detallan todos los cursos que realicé de esta plataforma, con los que aprendí conceptos y métodos para poder usarla en este proyecto.

Fue mientras realizaba los cursos de Neo4j cuando observé que la forma correcta de realizar un *recomendador* basado en texto—párrafos o frases—era basarse en vectores de dichos textos. Los vectores son representaciones numéricas multidimensionales de los textos. Pueden ir desde unas pocas dimensiones o componentes hasta cientos de miles. **Cuantas más componentes tengan los vectores, generalmente mejor es el algoritmo de recomendación.**

El último concepto que investigué, aunque superficialmente hasta el momento en el que debía comenzar su implementación, fueron los agentes. Lo primero fue comprender el concepto. La idea del **agente** es una entidad software que puede interactuar con su entorno, recopilando y utilizando datos para tareas concretas con el fin de cumplir ciertos objetivos de manera autónoma [17]. La clave es que puede interactuar y es autónomo.

Más concretamente, investigué a fondo los **agentes ReACT**. Estos son un tipo de agentes que además de tener la capacidad de ejecutar funciones predefinidas—como los agentes que llaman a funciones—razonan previamente con las capacidades de un LLM para luego realizar acciones concretas. Además, estos agentes pueden decidir que una acción no es lo suficientemente buena o es insuficiente para llegar al objetivo definido, y de este modo, ejecutar más de una acción o repetirla varias veces en caso de fallo. Por ejemplo una API puede fallar por un error temporal en la conexión.

Una vez comprendí todo lo anterior, estaba preparado para dar el salto a elegir una temática concreta para este trabajo. Y como en todas las aplicaciones de inteligencia artificial, el resultado final es completamente dependiente de los datos

con los que se entrena el modelo, o en mi caso, los datos que usará el modelo. **No se puede pretender que con malos datos se obtenga un buen resultado.** Es por ello, que tomé la decisión de buscar un *dataset* bueno a partir del cual construiría la aplicación. De este modo, se puede decir que es una *data-driven app*, es decir, una aplicación dirigida por los datos.

### 3.2 Búsqueda del dataset

Es en este punto donde entra en acción **Kaggle**, un sitio web en el que existen multitud de *datasets* de diversas índoles, temáticas y tamaños. Además, aparecen catalogados según la rama de IA a la que aplican (Classification, Computer Vision, NLP, ...). Entre otras métricas, para cada modelo se indica el tamaño del *dataset* y el número de ficheros de los que está compuesto, fecha de actualización, y usabilidad. Además, para cada *dataset* se pueden ver detalles, cómo la estructura del mismo, una previsualización de cada fichero que lo compone y estadísticas varias.

No quería usar un *dataset* pequeño, dado que temía no tener suficientes datos como para probar que el sistema recomendara correctamente, y me preocupaba también no tener suficiente información con la que realizar dichas recomendaciones. Es por ello por lo que añadí otro requisito al *dataset*: tenía que ser grande, preferiblemente de más de 1 GB.

El escogido fue **Amazon Book Reviews**, un *dataset* de más de 3 GB, dos ficheros CSV y tabular. Contiene información acerca de libros y reseñas. El fichero CSV de los libros contiene las siguientes columnas:

- Título del libro
- Descripción
- Autores
- Imagen
- Enlace de previsualización
- Editorial
- Fecha de publicación
- Enlace con información
- Categorías
- Contador de reseñas

De las reseñas, destacan entre otras las columnas precio, texto de la reseña, resumen de la reseña y fecha.

La usabilidad de este *dataset* es máxima, de 10.00 y provee una licencia CC0: Dominio Público [18]. Se pueden ver todos los detalles del *dataset* en la referencia [19].

Con todo lo anterior, parecía el *dataset* ideal para hacer este trabajo. Luego por tanto, la temática escogida para la aplicación sería “libros”. **Un recomendador de libros.**

### 3.3 Creación de una base de datos

Una vez tuve el *dataset* elegido y descargado, llegó el momento de poner en práctica los conocimientos adquiridos a través de los diversos cursos de Neo4j acerca de cómo cargar datos provenientes de un CSV en una base de datos.

El primer paso fue decidirme entre usar *Neo4j Desktop* o *AuraDB*, la versión de Neo4j online gratuita. El factor determinante, y a la vez limitante, fue que *AuraDB* no permite tener más de 200.000 nodos ni 400.000 relaciones en su versión gratuita. En cambio, el *dataset* tiene más de 210.000 libros, y 3.000.000 de reseñas. Por tanto, era **inviable usar *AuraDB***.

Una desventaja de Neo4j Desktop es, que en su versión gratuita, limita a un máximo de **4 hilos de paralelismo**. Esto, aunque pueda parecer suficiente, para BBDD grandes y medianas como en este caso, es un factor limitante tanto en la creación, como en la eliminación y a la hora de realizar consultas.

Una vez descargué e instalé *Neo4j Desktop*, probé a realizar un script en Python para cargar el *dataset* a la BBDD. Realicé distintas aproximaciones e intentos. Llegué a probar a introducir paralelismo para poder acelerar el proceso. En cambio, cada vez que ejecutaba la carga obtenía errores **de problemas en el formato de los datos**. Por ello, hice las oportunas correcciones manualmente en cada uno de los errores. Pero en cada ejecución se producía otro nuevo error en un lugar distinto.

Tras varios días, llegué a la conclusión de realizar un script en Python para corregir el *dataset* antes de intentar cargarlo. Dado que la inmensa mayoría se debían a **comillas dobles y simples que eran conflictivas** (triple comilla de apertura y doble de cierre, descripciones de libros entre comillas con comillas internas, etc.), y que el otro problema común se debía al carácter ‘\’ (barra inclinada) que suele tener el uso de escape de caracteres especiales, el código a desarrollar debía verificar solamente que las comillas estuvieran equilibradas, porque corrigiendo ese fallo se corregirían ambos. El otro problema era la aparición de comas (caracteres especiales para delimitar campos en el CSV) dentro de los textos— descripciones, reseñas, ...

El script desarrollado, ubicado en el repositorio en el directorio */data*, está compuesto de 3 funciones. De ellas destacan:

- *procesar\_linea*, que dada una línea, analiza cada carácter y detecta las distintas celdas que la componen. Para cada celda llama a la siguiente función:

- *procesar\_celda*: dada una celda, si esta está entrecomillada, elimina las comillas de apertura y cierre y sustituye las comillas internas dobles por simples (‘ por ’), y finalmente entrecomilla la celda de nuevo con comillas dobles.

Gracias a este script, el *dataset* se puede cargar correctamente.

Aun así, no conseguía realizar la carga de los datos en la BBDD en un tiempo razonable. Por ello, y con ayuda de herramientas de IA externas junto con los conocimientos adquiridos durante el desarrollo de cursos de **Cypher**, desarrollé un script en este lenguaje para cargar los datos. El fichero, en el directorio */utils*, se llama *load.cypher*.

Los nodos creados son:

- **Book**, para libros. Contiene datos como imagen, fecha de publicación, descripción, ...
- **User**, para usuarios que escriben reseñas.
- **Author**, para autores de libros. Contiene su nombre.
- **Genre**, para géneros. Contiene el nombre.
- **Publisher**, para editoriales. Contiene su nombre.
- **Review**, para cada reseña. Contiene propiedades como resumen, texto, utilidad, fecha de publicación, ...

En este script, se utiliza la biblioteca **APOC**, la cual sirve para implementar cientos de procedimientos y funciones que añade funcionalidad a Neo4j [20]. Además permite agilizar mediante optimización la carga de datos en comparación con su uso.

Finalmente, durante el desarrollo de la carga de datos, y debido a la necesidad de realizar diversas pruebas, desarrollé también **dos scripts auxiliares**, uno para gestionar la eliminación de todos los datos de la BBDD, y otro para generar un *dataset* reducido a partir del original y agilizar las pruebas.

Más adelante le daría a todo cierta orientación a objetos creando la clase *DBManager*.

### 3.4 Generación de los embeddings

Como se explica previamente en el punto 3.1, los *embeddings* son necesarios para poder realizar recomendaciones. Es la manera de **convertir texto en vectores** a los que se les puede luego aplicar algoritmos de cercanía para detectar por similitud qué textos se parecen entre sí.

El procedimiento del *recomendador* usando los *embeddings* puede ser principalmente, de dos maneras:

- Generar el *embedding* de un texto nuevo dado por el usuario, y se busca en la base de datos aquel que sea más similar, devolviendo el nodo (libro o reseña). Sería parecido a **buscar por contenido**.
- Comparar los *embeddings* ya creados y en base a ellos, recomendar textos similares. Por ejemplo, dado el título de un libro y teniendo su descripción y el *embedding* correspondiente, se pueden buscar libros similares a ese. Sería parecido a **recomendar por contenido**.

Como se ha dicho anteriormente, gracias a poder usar matemáticamente los vectores, se puede tanto buscar como recomendar teniendo en cuenta el contenido del nodo (libro o reseña).

Una vez razonada la importancia esencial de la creación de *embeddings*, llega el momento de buscar un modelo que los genere. Para ello, tuve en cuenta en la búsqueda la posibilidad del uso de diferentes idiomas. Es decir, teniendo el *dataset* en inglés (con las descripciones y las reseñas en inglés) posibilitar realizar recomendaciones o buscar libros en castellano u otro idioma. Por ello, **el modelo debía ser multilingüe**.

Tras buscar por diversos lugares, y teniendo en cuenta que en **Hugging Face** se ofrecen multitud de diversos modelos para distintos propósitos con ejemplos de uso, además de que ofrece una biblioteca para Python que simplifica el uso de estos modelos; decidí buscar directamente allí. En la referencia [21] se pueden ver los modelos disponibles en dicha plataforma.

Encontré la comparativa denominada **MTEB (Massive Text Embedding Benchmark) Leaderboard**, en la cual se ofrece una comparativa de distintos modelos de generación de *embeddings* para texto. Esta tabla además indica la memoria RAM necesaria en el sistema para poderlos ejecutar. Puede verse en [22].

Debido a las características de mi equipo de sobremesa, debido a que si se puede cargar el modelo completamente en la memoria dedicada de la tarjeta gráfica se reduce el tiempo de generación de *embeddings*, y debido a que quería tener **dos conjuntos de embeddings**; los generé con los modelos *sentence-transformers/all-MiniLM-L6-v2* y *dunzhang/stella\_en\_1.5B\_v5*. El primero de ellos genera vectores de 384 componentes, mientras que el segundo de 512 a 8192 dimensiones.

En la propia página del segundo modelo, se describe cómo **a partir de cierto punto, la mejoría del modelo con respecto a la cantidad de dimensiones se reduce drásticamente**, no compensando crear vectores de 8192 coordenadas [23]. Por ese motivo, los vectores generados con el segundo modelo son de 1536 componentes.

Los *embeddings* a generar eran los correspondientes a las más de 200.000 descripciones de libros, y aquellos que representan los resúmenes de las más de

3.000.000 reseñas junto con los propios textos de dichas reseñas. Con lo anterior, el total aproximado ronda los **6.200.000 textos a procesar**.

Dado que el *dataset* es extremadamente grande como para generar los *embeddings* cada vez que sea preciso disponer de ellos, tuve que guardarlos en la BBDD para mejorar el rendimiento. Para ello desarrollé el fichero `/models/embedding_manager.py` y en `DBManager` agregué la función `generate_embeddings_for` y sus auxiliares. Con ello, se genera un lote de *embeddings* de texto que luego se guarda en la BBDD. Para generar y guardar ambos conjuntos de *embeddings*, el tiempo de ejecución necesario fue considerable. El primer modelo tardó aproximadamente 16 horas, mientras que el segundo necesitó cerca de **48 horas**. Todo esto con una NVIDIA RTX 4070, una tarjeta gráfica que dentro del mundo doméstico es de gama alta en el momento actual.

Tras generar los *embeddings*—en dos BBDD separadas para no mezclarlos—hice los correspondientes volcados para tener puntos de restauración de la BBDD por seguridad. El primer fichero (que contiene toda la BBDD) ocupa 36 GB, mientras que el segundo, **56,7 GB**. El aumento desde los 3 GB originales se debe principalmente al almacenamiento de los vectores.

Para trabajar con los vectores producidos, se recomienda usar bases de datos vectoriales. En cambio, las BBDD orientadas a grafos ofrecen la posibilidad de poder conectar la información fácilmente, como se ha desarrollado en este proyecto. Neo4j soporta la rápida búsqueda a través de vectores mediante los **índices vectoriales**, uniendo las ventajas de ambos tipos de BBDD. La diferencia entre usar índices vectoriales para la búsqueda de similitud mediante *embeddings* y no usarlos es drástica. Empíricamente la mejoría es del orden de minutos.

Entrando más en el campo matemático, la búsqueda o recomendación con los *embeddings* generados funciona de la siguiente manera. Un vector de tres coordenadas representa un punto en el espacio tridimensional. Con varios puntos, se puede calcular la distancia entre ellos y otro dado, obteniendo el más cercano. Si se tienen más de tres componentes, se habla de un espacio multidimensional, pero la forma de calcular la distancia es generalizable.

Otra forma de calcular similitud, más común, es mediante el uso del coseno. La **similitud coseno** se calcula con el valor del coseno del ángulo formado entre los dos vectores dados (puntos del hiperespacio) y el origen de coordenadas (el vector nulo). La imagen de la función está comprendida en el intervalo  $[-1, 1]$ , siendo 1 cuando la similitud es máxima, es decir, cuando ambos vectores tienen la misma dirección.



### 3.5 Creación de agentes REACT con herramientas RAG

Una vez que tenía la BBDD con el *dataset* cargado y los *embeddings* generados, llega el momento de introducir agentes. Y con los conocimientos adquiridos durante la fase de aprendizaje, deduje que lo primero era desarrollar herramientas para poder crear un agente RAG.

Por ello, desarrollé distintas funciones en el fichero `/agents/tolos/rag_tools.py` para acceder a la BBDD y obtener información. Son las siguientes:

- **Recomendar libros parecidos.** Dado un texto y la cantidad de libros a recomendar, retorna los k libros más parecidos al texto dado. El texto puede ser el título de un libro, en cuyo caso se recomiendan libros similares a ese basándose en los *embeddings* de las descripciones. Si no coincide con ningún título, se genera el *embedding* de ese texto y se recomiendan libros en función de los *embeddings* de los títulos y de las descripciones presentes en la base de datos. Se devuelven los títulos de los libros junto con el valor de similitud.
- **Recomendar por género.** Dado un título, se recomiendan k libros pertenecientes al mismo género literario. En caso de disponer de descripciones, se utilizan para la recomendación. Se devuelven los títulos de los libros junto con el valor de similitud.
- **Recomendar por autor.** Análogo a “recomendar por género”, pero con autores.
- **Obtener descripción de libro.** Dado un libro, retorna la descripción del mismo.
- **Obtener información de libros.** Dada una lista de libros, retorna un diccionario donde las claves son los títulos de los libros y los valores son “diccionarios” conteniendo la información de cada libro. Incluye descripción, autor, género, fecha de publicación y URL de la imagen.
- **Obtener reseñas de libro.** Dado el título de un libro, retorna todas las reseñas que comentan el libro.
- **Recomendar libros por reseñas.** Dado un texto, busca reseñas cuyo contenido sea similar (usando los *embeddings*) y retorna los libros correspondientes a las k reseñas más parecidas. Cabe destacar que esta es la función más lenta al ser la BBDD tan grande que no es posible cargarla en memoria principal. Una llamada a esta función puede tardar más de 5 minutos en retornar una respuesta.
- **Obtener libros de un autor.** Dado un autor, busca y devuelve los libros sobre los que tiene autoría. Admite un parámetro limitador máximo para los libros devueltos.

Todas las funciones deben estar correctamente documentadas para que el agente las pueda utilizar y comprenda qué es lo que hacen, qué parámetros necesitan, cómo funcionan y qué devuelven tras su ejecución. Esto se hace en Python mediante el uso de las llamadas **docstrings**, cadenas de texto entrecomilladas con triple comilla.

Como se puede intuir, son pocas funciones como para hacer un buen sistema de recomendaciones. Pero en cambio, basta para poder tener una demo de agente que utiliza las herramientas correctamente para acceder a la BBDD y responder a lo que el usuario pide.

Tras probar una a una todas las funciones y ver su correcto funcionamiento, llegó el momento de **investigar más a fondo cómo funcionaban los agentes**. Durante esa investigación me enfoqué en LlamaIndex y en LangChain. Tanto uno como el otro ofrecen bibliotecas en Python para trabajar con agentes. Y tras ver vídeos en YouTube sobre el funcionamiento interno, junto con ejemplos explicados en distintos sitios web, además de probar códigos en Google Colaboratory—un servicio alojado de Jupyter Notebook que proporciona acceso gratuito a recursos informáticos como capacidad de cómputo, GPUs y TPUs [24]—vi que todo agente necesita una conexión a una API de un LLM.

Entonces comenzó la investigación de cómo poder exponer una API de un LLM en local. Encontré **Ollama**, que permite ejecutar múltiples LLMs en local. Ollama proporciona un servidor y una interfaz por línea de comandos para ejecutar los modelos deseados. Además incluye un repositorio online con distintos modelos, facilitando todo el proceso.

Probé distintos modelos mediante la interfaz por línea de comandos, grandes y pequeños, y acabé quedándome con tres:

- **Llama3.2** para respuestas rápidas, que al ser mediano de tamaño (2GB) es bastante rápido en mi equipo.
- **Mixtral:8x7b** que es un modelo grande (26GB) que no es excesivamente lento, y proporciona respuestas mucho más finas que el anterior.
- **Llama3.3** que con 43GB da las mejores respuestas que he podido probar en mi equipo, que tiene 64GB de RAM. En comparación, es bastante más lento que los otros dos modelos, pero la calidad de las respuestas es superior.

De este modo, usaría llama3.2 para probar rápidamente que el código se ejecuta correctamente, y los otros dos LLMs para realizar pruebas de casos de uso, con ejemplos reales y centrándome más en la calidad de las respuestas que en el propio funcionamiento.

Luego comencé la implementación de un **agente ReACT**. Primero con LangChain, pero tras varios días sin conseguir eliminar avisos de métodos obsoletos, y no

siendo capaz de mantener modularizadas las herramientas de RAG desarrolladas anteriormente, me decanté por probar a hacerlo con LlamaIndex. Tras varias versiones de código, conseguí un agente completamente funcional. Este agente está en la clase *RagAgent*.

Durante el intento con LangChain, desarrollé un **prompt** para el agente, que es un texto que describe la forma de actuar del agente, para que el LLM comprenda qué debe hacer. En LangChain es necesario indicar un prompt para cada agente, pero en LlamaIndex no lo es.

Con las pruebas, observé el **proceso de ejecución del agente**. Comienza con la inicialización de este, indicándole el LLM que usará, así como las herramientas (lista de funciones) y la cantidad máxima de iteraciones. Luego espera la entrada del usuario, que puede ser de cualquier índole. Supongamos que es una petición para algo para lo que el agente está capacitado—tiene funciones para ello. Una vez se envía, el agente recibe la petición y razona qué es lo que debe hacer para responder, qué herramientas ha de utilizar (una, varias, o ninguna si puede responder directamente). Tras el razonamiento, actúa llamando a alguna función. Si con la respuesta obtenida razona que no dispone de suficientes datos como para responder al usuario, vuelve a elegir una herramienta para obtener más información. Cuando considera que puede responder al usuario, elabora una respuesta con todos los datos proporcionados por las herramientas.

Es importante para una buena experiencia del usuario que el agente disponga de **memoria**, es decir, tenga el contexto de la conversación que ha mantenido hasta el momento con el usuario. Esto permite al usuario actuar de forma más natural obviando información en los mensajes. Por ejemplo, si acaba de preguntar por un libro y el agente le devuelve “... El mejor libro para que leas es El Señor de las Moscas”, el usuario puede preguntar directamente “¿de qué trata?” en lugar de tener que especificar “¿de qué trata El Señor de las Moscas?”. Con LangChain se debe especificar el funcionamiento de la memoria y el contexto de la conversación, mientras que con LlamaIndex está implícito.

Un pequeño problema que tuve fue la gestión de la lentitud al generar los LLMs más grandes las respuestas necesarias para que el agente piense y trabaje, dado que obtenía **errores de tipo timeout**, fin de tiempo de espera. Al cortarse la ejecución por tardar demasiado tiempo en llegar al agente la respuesta del LLM, era un error crítico. Después de cierta investigación, lo solventé aumentando el tiempo límite de espera de la respuesta solicitada a Ollama a 7 minutos. Está en la inicialización del atributo *llm* de *RagAgent*.

### 3.6 Desarrollo de la interfaz web

Con lo anterior desarrollado, ya disponía de un sistema *recomendador* que por medio de un agente de RAG, podía recomendar a través de una conversación con un usuario libros según la BBDD. El último paso consistía en dar un **formato atractivo que añadiera el mínimo código** posible para la interfaz.

Tras investigar en distintos lugares y repositorios, encontré que en varios de ellos cuya interacción con el usuario se basaba en texto conversacional—*chatbots*—, utilizaban una biblioteca en Python denominada Streamlit. Todos los ejemplos que vi en GitHub proponían **interfaces elegantes, simples e intuitivas** que además, no añadían complejidad al código. Y por eso opté por usar Streamlit para la interfaz. También valoré realizar la interfaz de usuario mediante una aplicación de escritorio tradicional con ventanas. Mas la idea fue descartada al ver en la búsqueda previamente mencionada que prácticamente en ningún proyecto parecido se llevaba a cabo.

**Streamlit genera tanto el *frontend* como un servidor web** para poder acceder a la interfaz de usuario de forma transparente al desarrollador. Es tal el grado de simplicidad que todo el código del *frontend* está en el fichero `/view/ui.py`, que documentado y sin usar *comprehensions*—una forma de comprimir varias líneas de código en Python en una sola—**no llega a contener 70 líneas**. Aunque el número de líneas no es una métrica perfecta de un código, sí que puede ser usado para dar una idea de la extensión del mismo.

Es tan simple el código en Streamlit porque incluye varios casos de uso típicos de interfaces con IA generativa, como los *chatbots*. De modo que simplemente indicando que va a mostrar un chat, y en cada mensaje especificando quién es el autor del mismo con palabras como “user” y “assistant”, representa una conversación con formato visualmente agradable y con iconos intuitivos de quién ha emitido cada mensaje.

Además, incluye concurrencia en la ejecución mediante el uso de paralelismo. Esto **ayuda al usuario a no impacientarse** dado que está recibiendo continuamente realimentación de lo que está realizando el sistema. Concretamente, al estar ejecutando todo en local, las respuestas del LLM pueden tardar, además de que es probable que se requiera más de una invocación para los distintos pensamientos y acciones del agente. Por ello, y como se puede observar en la figura 3.6.1, aparece en la esquina superior derecha “RUNNING” con una animación, indicando que el agente está trabajando. Todo esto **sin necesidad de especificar nada en el código**.

Al usar Streamlit, la forma de ejecutar el fichero de Python varía notablemente. Típicamente, una aplicación en Python se ejecuta haciendo una llamada al intérprete, por ejemplo, para ejecutar el fichero `main.py`, que acostumbra a ser el punto de entrada a las aplicaciones desarrolladas en este lenguaje, se debe

ejecutar en la terminal “*python3 main.py*”. En cambio, si la aplicación utiliza Streamlit, se debe **ejecutar con el comando “*streamlit run main.py*”**.

Lo explicado en el párrafo anterior fue un error que tardé en solucionar un tiempo, al no comprender por qué no se ejecutaba correctamente el código al hacer clic en el botón de ejecutar presente en el editor (Visual Studio Code) tras implementar el *frontend* con Streamlit; pues el mensaje de error generado es bastante largo y menciona problemas con hilos de ejecución.

Tras solucionar ese error, realicé varias pruebas. Y tras solventar pequeños errores, finalmente llegué a una versión completamente funcional, como se puede observar en la figura 3.6.1.

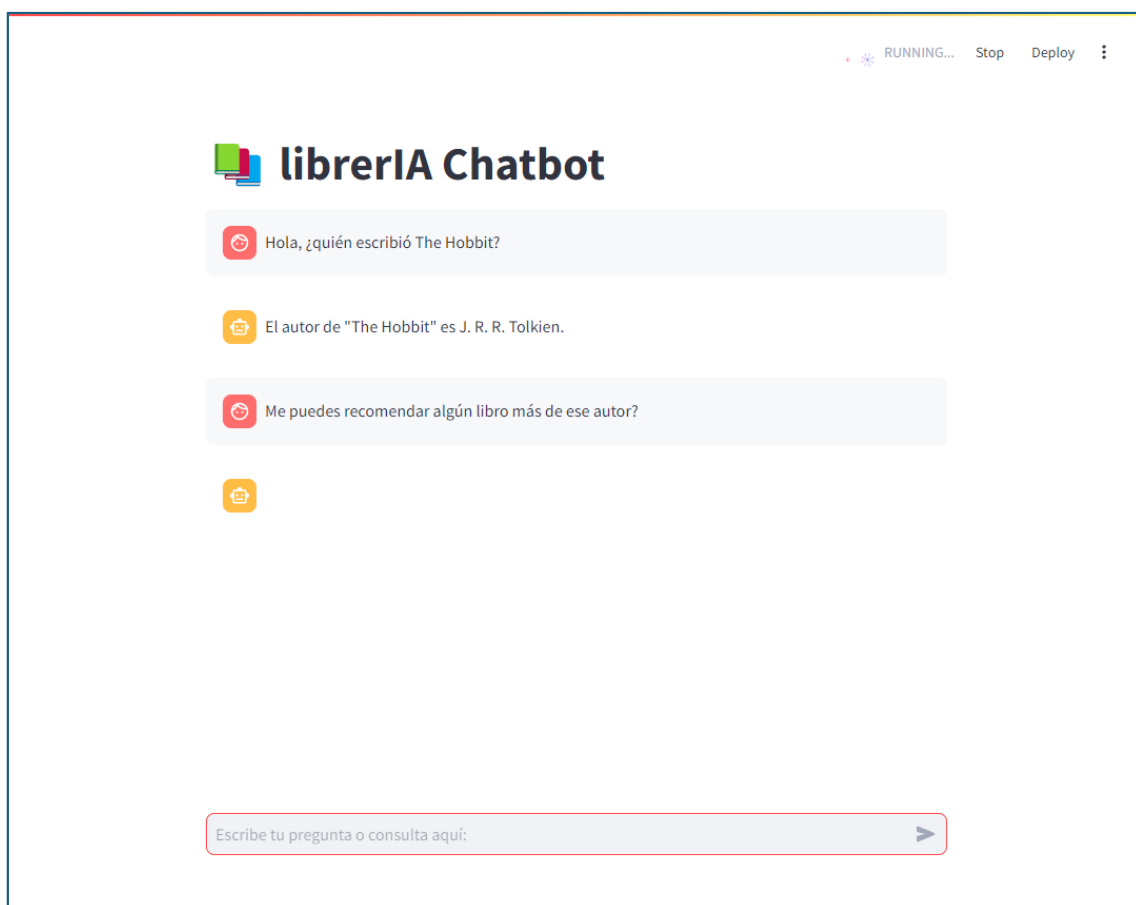


Figura 3.6.1 - Interfaz de la aplicación. Chat de Streamlit. Elaboración propia

### 3.7 Integración de dictado por voz. Whisper

Cuando tuve desarrollada la primera versión básica de la interfaz, vi que aún disponía de suficiente tiempo como para intentar **agregar algún tipo de funcionalidad nueva**. Entre los diferentes aspectos que valoré estaba la integración de un dictado por voz. Recordé que OpenAI había lanzado varios modelos de transcripciones multilingüe bastante potentes—ya que lo había probado previamente—y que podía ejecutar en local, al igual que todo lo demás. Los habían denominado Whisper, *susurro* en inglés.

La idea surge dado que, aunque multitud de sistemas operativos ofrecen servicios de dictado por voz, la mayoría de estos no funcionan sin conexión a Internet, luego no se ejecutan completamente en local. Además quería integrar toda la funcionalidad dentro de la misma aplicación para no depender del sistema operativo del usuario final.

El principal problema era que **todos los ejemplos oficiales venían hechos para trabajar con ficheros de audio** pregrabados. Es decir, no había ningún ejemplo que yo conociera previamente de uso de Whisper con transcripciones en tiempo real. Yo imaginaba que, si era posible trabajar con ficheros completos, también se podría trabajar con audio en directo. Por ello, nuevamente tuve que investigar acerca de esta tecnología.

Di con un repositorio en GitHub, disponible en la referencia [25], en el que se usa un modelo de Whisper a elegir entre diversos tamaños e idiomas (inglés o multilingüe), que con acceso al micrófono del dispositivo en el que se ejecuta, **transcribe en la terminal lo que recibe el micrófono más o menos en tiempo real**. Probándolo en mi PC, analicé el código fuente—que se compone de un único fichero de Python—y decidí lanzarme a implementarlo como una característica adicional a mi aplicación *recomendadora*.

Para ello, comencé desarrollando el fichero `/models/transcript_manager.py`, donde se ubica la clase `TranscriptManager`, que contiene: un método para comenzar la escucha, otro método para finalizarla y un tercero para obtener la transcripción. Estos son los métodos más importantes, y los que luego usaría en la interfaz.

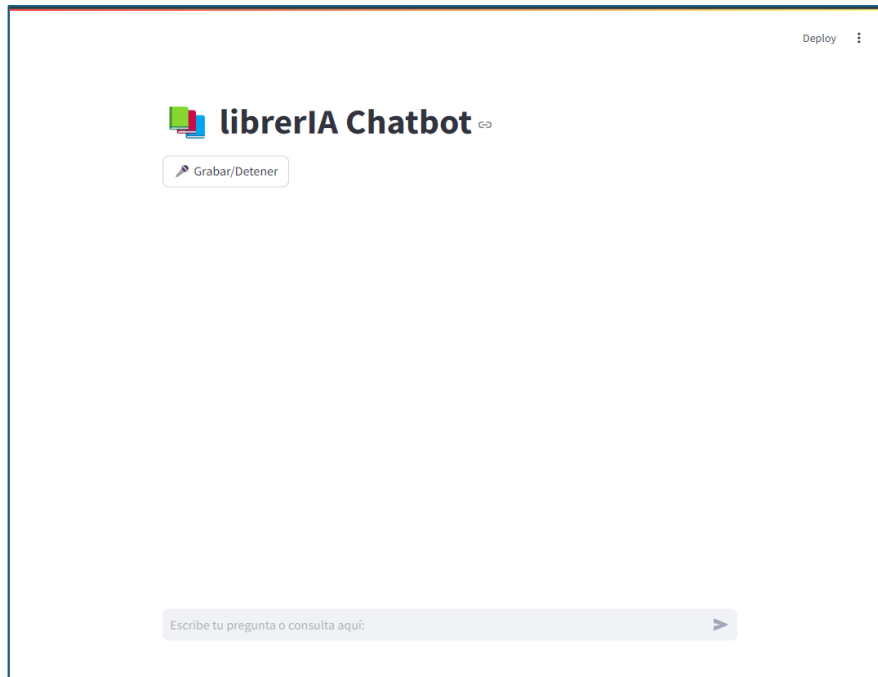


Figura 3.7.1 - Interfaz de la aplicación con botón para transcribir. Elaboración propia

En la interfaz, típicamente se pondría un icono de un micrófono en la zona de escritura de mensajes del usuario—la zona inferior de la figura 3.6.1. Sin embargo, dado que ese elemento viene predefinido en Streamlit, no podía editarlo para añadirle un micrófono. Y dado que el tiempo comenzaba a escasear, decidí aplicar una de las soluciones más sencillas (y quizá no demasiado oportuna): añadir un botón. El botón se situaría en la zona superior, no siendo explícitamente intuitivo al igual que el resto del sistema. Se observa en la figura 3.7.1.

### 3.8 Realización de pruebas de funcionamiento

Pese a que fui desarrollando pruebas de funcionamiento del sistema conforme iba desarrollando el proyecto, al final de este, dediqué un tiempo específico a probarlo en mayor profundidad.

Algunas de estas pruebas fueron, por ejemplo, la búsqueda de resultados específicos. Para ello, busqué en la BBDD un libro concreto, y realicé consultas para obtener los 5 libros con mayor similitud al dado. De este modo sabría los resultados que debía esperar del sistema. A continuación, preguntaría al *chatbot* por recomendaciones de ese mismo libro para ver si las respuestas coincidían. La prueba en sí consiste **en observar hasta qué nivel de abstracción en el mensaje del usuario es capaz el agente de comprender** qué es lo que se le está pidiendo y qué herramientas ha de utilizar. Los resultados de este tipo de pruebas fueron los

esperados. Con LLMs grandes ejecutándose en Ollama, el agente es mucho más preciso, capaz y potente que con el modelo mediano (llama3.2).

Otro tipo de pruebas que realicé fueron para comprobar el sistema de reconocimiento de voz. Para ello, escogí diferentes tamaños del modelo Whisper y probé a utilizar la funcionalidad. Los resultados fueron que los modelos más grandes tardan más tiempo en cargarse en memoria, haciendo que el usuario deba esperar más tiempo para poder comenzar a hablar. En cambio, las versiones de menor tamaño no son capaces de reconocer correctamente todas las palabras y expresiones. La conclusión fue que un modelo de tamaño mediano contenía el **equilibrio entre tiempo de carga y calidad del procesamiento**. Se debe tener en cuenta que no es posible precargar el modelo en memoria ya que tanto la BBDD como los LLMs ejecutándose en Ollama requieren gran cantidad de memoria principal (RAM), luego no abunda la memoria libre disponible.

Durante la realización de las pruebas de funcionamiento anteriores, noté que faltaban herramientas de tipo RAG para que el agente pudiera funcionar mejor. Y como no eran demasiado complejas y dotarían al agente de mayores capacidades, decidí implementarlas. Estas están entre las mencionadas en el punto 3.5.



## 4 Resultados y conclusiones

En este cuarto apartado se presenta un análisis detallado de los resultados obtenidos durante la realización del proyecto, evaluando su alineación con los objetivos planteados inicialmente. Asimismo, se exponen las principales conclusiones derivadas del trabajo, destacando las lecciones aprendidas, las limitaciones encontradas y las posibles líneas de mejora o continuidad para futuros desarrollos.

### 4.1 Análisis de resultados

En los siguientes párrafos se elaboran ejemplos ilustrados con figuras de distintas situaciones de uso de la aplicación y las respuestas que proporciona el sistema para cada una de ellas.

#### 4.1.1 Recomendación de libros genérica con LLM grande

En este primer ejemplo, se usa el modelo LLM llama3.3, el de mayor tamaño de los probados. Se puede observar en la figura 4.1.1 la respuesta del agente a la pregunta hecha por el usuario.



Figura 4.1.1 – Recomendación de libros similares a uno dado. Elaboración propia

Destaca de la respuesta del agente el uso de herramientas auxiliares a la recomendación en sí, en este caso, para proporcionar una descripción junto con el título del libro. Se puede observar a través de los logs que el agente muestra por terminal, visibles en la figura 4.1.2, siendo las acciones, las herramientas que el agente utiliza; las observaciones, los resultados de ejecutar dichas acciones; y los pensamientos, el razonamiento que realiza el agente previo a realizar alguna acción.

```

> Running step d84f5e47-e35b-4a18-ab9d-db5151b1f8a8. Step input: Hola. Ayer terminé de leer The Hobbit. Me puedes hacer alguna recomendación sobre qué puedo leer ahora?
Thought: The current language of the user is Spanish. I need to use a tool to help me answer the question. To recommend books, I can use the recommendSimilarBooks tool with the title of the book the user just finished reading.
Action: recommendSimilarBooks
Action Input: {'input_text': 'The Hobbit', 'top_k': 5}
Observation: [{"The Hobbit or There and Back Again", 0.9999999996744069}, {"The Hobbit", 0.9999999996744069}, {"Firstborn (Dragonlance Elven Nations, Vol 1)", 0.5352136450055002}, {"Wind in the Wires: A Golden Era of Flight, 1909-1939", 0.5280832429225132}, {"Space-Time Structure", 0.5269377941639468}]
> Running step 35f4ba08-b4fc-43eb-96e5-e66f5190c357. Step input: None
Thought: The recommendSimilarBooks tool returned a list of books with their similarity scores. However, the top two results are the same book the user just finished reading, which is not helpful for recommendations. I need to remove these duplicates and return the next most similar books.
Action: recommendSimilarBooks
Action Input: {'input_text': 'The Hobbit', 'top_k': 10}
Observation: [{"The Hobbit", 0.9999999996744069}, {"The Hobbit or There and Back Again", 0.9999999996744069}, {"Firstborn (Dragonlance Elven Nations, Vol 1)", 0.5352136450055002}, {"Wind in the Wires: A Golden Era of Flight, 1909-1939", 0.5280832429225132}, {"Space-time structure", 0.5269377941639468}, {"Space-Time Structure", 0.5269377941639468}, {"Unsung Heroes of The Lord of the Rings: From the Page to the Screen", 0.5267634040009346}, {"The Disdainful Marquis, and The Abandoned Bride (Signet Regency Romance)", 0.5190942456678747}, {"The Decline and Fall of the Roman Empire (volume1)", 0.5087712814136287}, {"The decline and fall of the Roman empire (Everyman's Library)", 0.5087712814136287}]
> Running step b16b41e5-946c-4b84-a26e-07b576c930f4. Step input: None
Thought: I can answer without using any more tools. I'll use the user's language to answer. Based on the information provided by the recommendSimilarBooks tool and the description of 'Firstborn (Dragonlance Elven Nations, Vol 1)', I can recommend this book as it has a high similarity score with 'The Hobbit'.
Answer: Te recomiendo leer 'Firstborn (Dragonlance Elven Nations, Vol 1)' ya que tiene un alto grado de similitud con 'The Hobbit'. Esta es la primera entrega de una trilogía clásica publicada en 1991 y ahora disponible en una edición de mercado masivo.

```

Figura 4.1.2 - Logs del agente recomendando libros similares a uno dado. Elaboración propia

#### 4.1.2 Recomendación de libros genérica con LLM mediano

En este ejemplo de uso se ilustra el motivo por el cuál es preferible utilizar un LLM grande frente a uno mediano, especialmente cuando las herramientas pueden no ser coherentes o completamente consistentes. En este dataset existe el libro *The Hobbit* pero no está especificado su autor. Por ello, al realizar la misma consulta que en el apartado previo, la respuesta es drásticamente distinta. Puede verse en la figura 4.1.3. En cambio, el proceso de razonamiento que ha llevado a cabo el agente ha sido mucho más complejo que el anterior para llegar, en principio, a una conclusión mucho peor.

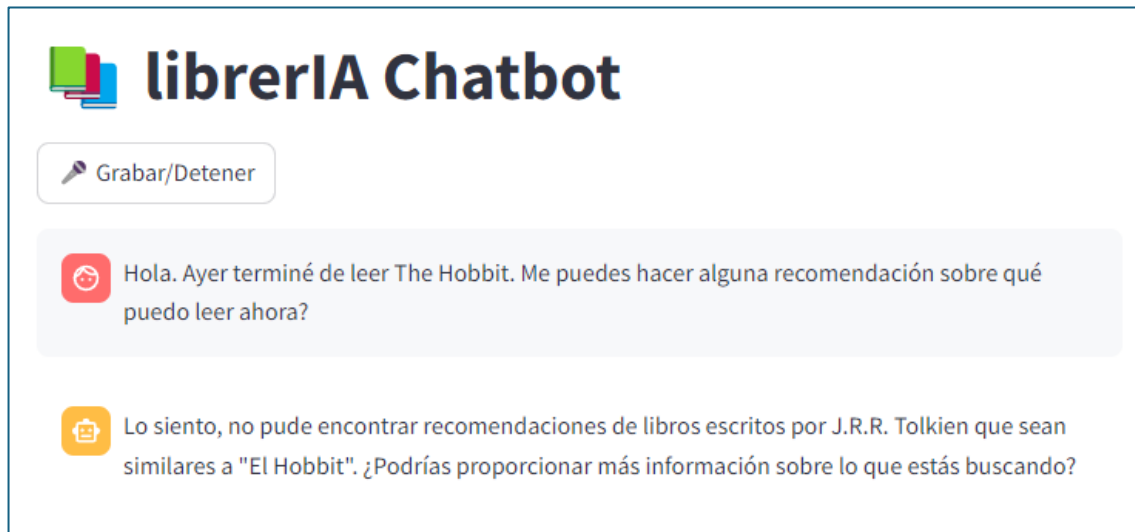


Figura 4.1.3 – Recomendación de libros similares a uno dado. Elaboración propia

Repitiendo exactamente la misma consulta una vez más, se obtiene una respuesta peculiarmente distinta. Esto se debe a la **temperatura**, un parámetro que controla el grado de aleatoriedad del LLM. La siguiente ejecución generó un error por alcanzar el número máximo de llamadas al LLM dado que el agente entró en bucle. Véase la figura 4.1.4.

En un tercer intento, el agente trató de obtener información adicional acerca de los libros a recomendar provistos por la herramienta, pero indicó mal la entrada y no

obtuvo respuestas. Por ello, acabó utilizando su propio conocimiento para recomendar autores de fantasía en lugar de libros, que fue lo que se le solicitó.

```
> Running step 46201693-1f80-40db-886f-c850718c4c46. Step input: Hola. Ayer terminé de leer The Hobbit. Me puedes hacer alguna recomendación sobre qué puedo leer ahora?
Thought: The current language of the user is: español. I need to use a tool to help me answer the question.
Action: recommendSimilarBooks
Action Input: {'input_text': 'The Hobbit', 'top_k': 5, 'title_embedding_property': 'title_embedding', 'description_embedding_property': 'description_embedding'}
Observation: [{'The Hobbit or There and Back Again', 0.9999999996744069}, {'The Hobbit', 0.9999999996744069}, {'Firstborn (Dragonlance Elven Nations, Vol 1)', 0.5352136450055002}, {'Wind in the Wires: A Golden Era of Flight, 1909-1939', 0.5280832429225132}, {'Space-Time Structure', 0.5269377941639468}]
> Running step 4646c281-2ec5-4d4a-a6c0-e8c0d7a76a9e. Step input: None
Thought: The user has been provided with a list of book recommendations similar to "The Hobbit". I need to analyze the results and provide further guidance.
Action: getBooksInfo
Action Input: {'books': ["'The Hobbit or There and Back Again'", "'Firstborn (Dragonlance Elven Nations, Vol 1)'", "'Wind in the Wires: A Golden Era of Flight, 1909-1939'", "'Space-Time Structure'"]}
Observation: {}
> Running step 1727ccb9-33a7-42b5-9f9b-fce14360cc90. Step input: None
Thought: The tool did not provide any additional information about the recommended books. I need to consider other factors to make a recommendation.
Action: recommendSomeGenres
Action Input: {'book_title': 'The Hobbit', 'top_k': 5, 'description_embedding_property': 'description_embedding'}
Observation: [{"The Hobbit or There and Back Again", 1.0}, {"We Were There at the Boston Tea Party (We Were There Books)", 0.49376191619535237}, {"More Wise Men of Helm and Their Merry Tales", 0.48760120188051447}, {"The Wise Men of Helm and Their Merry Tales", 0.48760120188051447}, {"We Were There at the Driving of the Golden Spike (We Were There, 31)", 0.4803776970988382}]
> Running step e95da7c5-82d7-47f8-945f-8b20e14aab2. Step input: None
Thought: The tool has recommended books in the same genre as "The Hobbit". I need to consider the user's interest and reading preferences.
Action: getBooksFromAuthor
Action Input: {'author': 'J.R.R. Tolkien'}
Observation: {}
> Running step f57d4f35-8693-496b-9dda-e9c9a9ae437c1. Step input: None
Thought: J.R.R. Tolkien is a well-known author, but it seems that none of his books are available in the database or not relevant to the user's interests. I need to consider other authors and genres.
Action: recommendBooksByReviews
Action Input: {'review': 'I enjoyed "The Hobbit" by J.R.R. Tolkien', 'h': 5}
Observation: {}
Action: recommendSomeGenres
Action Input: {'book_title': 'The Hobbit', 'top_k': 5, 'description_embedding_property': 'description_embedding'}
Observation: [{"The Hobbit or There and Back Again", 1.0}, {"We Were There at the Boston Tea Party (We Were There Books)", 0.49376191619535237}, {"More Wise Men of Helm and Their Merry Tales", 0.48760120188051447}, {"The Wise Men of Helm and Their Merry Tales", 0.48760120188051447}, {"We Were There at the Driving of the Golden Spike (We Were There, 31)", 0.4803776970988382}]
> Running step e95da7c5-82d7-47f8-945f-8b20e14aab2. Step input: None
Thought: The tool has recommended books in the same genre as "The Hobbit". I need to consider the user's interest and reading preferences.
Action: getBooksFromAuthor
Action Input: {'author': 'J.R.R. Tolkien'}
Observation: {}
> Running step f57d4f35-8693-496b-9dda-e9c9a9ae437c1. Step input: None
Thought: J.R.R. Tolkien is a well-known author, but it seems that none of his books are available in the database or not relevant to the user's interests. I need to consider other authors and genres.
Action: recommendBooksByReviews
Action Input: {'review': 'I enjoyed "The Hobbit" by J.R.R. Tolkien', 'h': 5}
Observation: {}
```

Figura 4.1.4 - Logs del agente al recomendar un libro y quedarse en bucle. Elaboración propia

Cabe destacar que la velocidad a la hora de generar las respuestas supera con creces aquellos modelos de mayor tamaño. En cambio la calidad de estas es mucho más cuestionable. En líneas generales, puede llegar a ser equivalente ejecutar la consulta con un modelo grande y ejecutar varias veces la consulta con un modelo mediano hasta obtener una respuesta satisfactoria.

### 4.1.3 Resolución de dudas acerca de un libro dado

Para este ejemplo, tomaremos el libro “Pyrography Designs”, que contiene ilustraciones artísticas. Al no ser un libro tradicional de lectura, es muy probable que el LLM no contenga información suficiente como para generar una respuesta únicamente basada en su conocimiento previo. Lo cual es beneficioso ya que usa los datos de los que dispone, evitando así errores de datos obsoletos o alucinaciones. En esta conversación se pregunta información también de otro libro, viendo como el historial de esta no desvía la atención del agente. La figura 4.1.6 muestra la conversación, mientras que la figura 4.1.5 muestra la información de estos dos ejemplares en la BBDD.

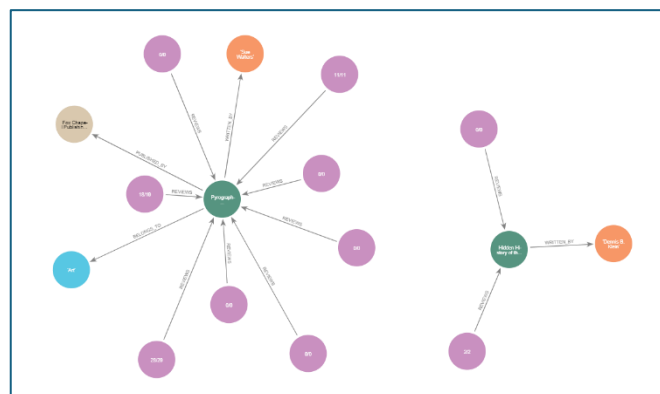








Figura 4.1.5 - Datos en la BBDD orientada a grafos y sus relaciones. Elaboración propia


 **librerIA Chatbot** 


Grabar/Detener 


 qué contiene Pyrography Designs?


 Esta libreta de Pyrography Designs contiene 30 ilustraciones de vida silvestre norteamericana y 10 diseños para bordes que se pueden utilizar en proyectos de quemado con carbón. Se proporcionan diseños grandes y listos para usar tanto en patrones líneales como tonales. El autor incluye consejos sobre cómo transferir patrones, además de consejos sobre segmentar y manipular las imágenes para crear tus propios diseños personalizados.


 Quién escribió "Hidden History of the Kovno Ghetto"?


 Dennis B. Klein wrote "Hidden History of the Kovno Ghetto". The book is a visual and documentary record that presents a compelling view of Jewish life, loss, survival, and defiance during the Holocaust.

 A qué género pertenece Pyrography Designs?

 Pyrography Designs pertenece al género de arte.

 Qué opina la gente de Hidden History of the Kovno Ghetto?

 La gente opina muy bien de "Hidden History of the Kovno Ghetto", considerándolo una obra importante y emocional que ofrece una visión profunda sobre la historia del ghetto durante la Segunda Guerra Mundial.

 Qué editorial está detrás de Pyrography Designs?


 La editorial detrás de Pyrography Designs es la Fox Chapel Publishing Company Incorporated.

Figura 4.1.6 - Conversación para obtener información de dos ejemplares. Elaboración propia

#### 4.1.4 Recomendación de libros similares a uno dado

En este ejemplo, el mensaje del usuario da más información de la estrictamente necesaria al modelo según las herramientas de las que dispone. Pero aun así, responde con dos libros de manera correcta. Se observa en la figura 4.1.7.

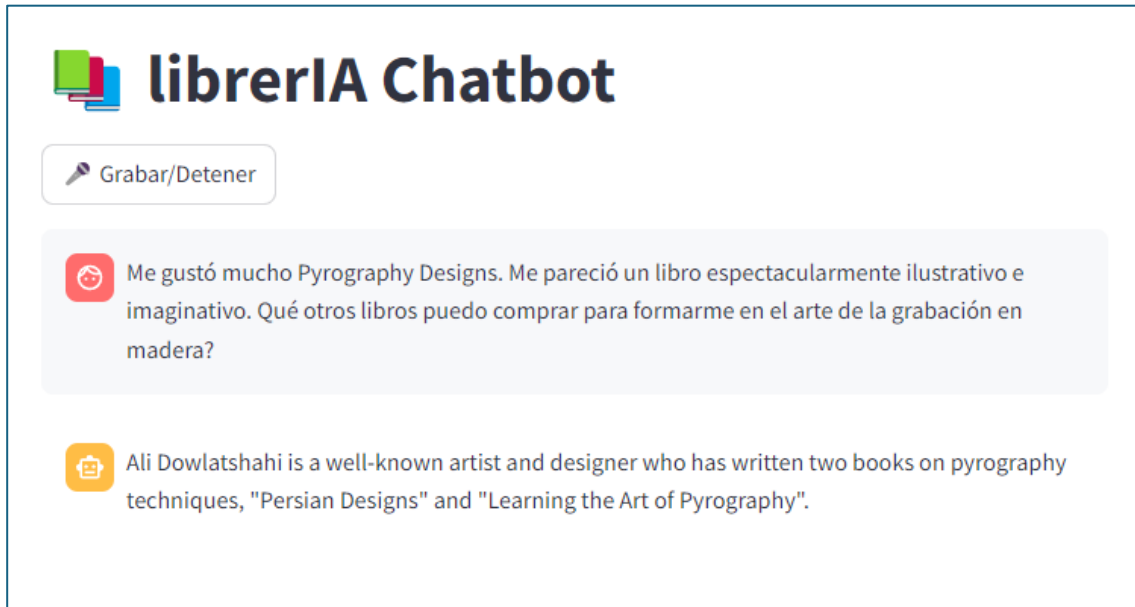


Figura 4.1.7 - Recomendación de libros a partir de uno dado. Elaboración propia

#### 4.1.5 Recomendación de libros en inglés

En este ejemplo, solicitamos al chatbot que nos recomiende un libro que contenga diseños y patrones decorativos y tradicionales. Pero realizamos dicha consulta en inglés. Se espera que el chatbot responda en el idioma del usuario, en inglés. En la figura 4.1.8 se puede ver la respuesta del agente, y en la figura 4.1.9 el proceso que ha producido dicha respuesta.

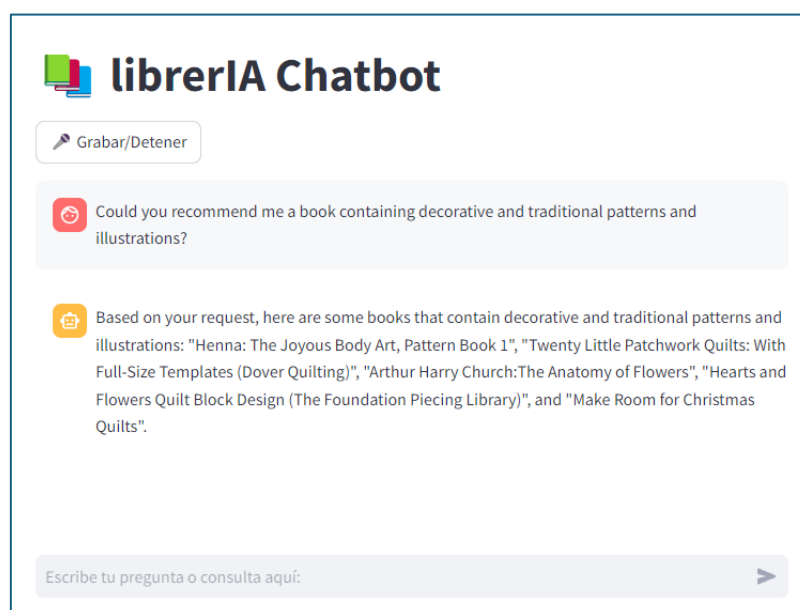


Figura 4.1.8 - Recomendación de libros usando inglés. Elaboración propia

```
> Running step e1a8d434-d396-4712-bfcf-f2689954b8a2. Step input: Could you recommend me a book containing decorative and traditional patterns and illustrations from regions nearby Iran?
Thought: The user is looking for a book with decorative and traditional patterns and illustrations from regions nearby Iran. To find such a book, I need to use a tool that can provide recommendations based on the content of the book.
Action: recommendSimilarBooks
Action Input: {'input_text': 'decorative and traditional patterns and illustrations from regions nearby Iran', 'top_k': 5}
Observation: [{"Natural history and antiquities of Selborne", 0.8994751657846841}, {"Strange customs of courtship and marriage", 0.8982065556276346}, {"Real estate principles and practices", 0.8948227232500779}, {"Permanent magnet design and application handbook", 0.894540017082598}, {"A pictorial history of magic and the supernatural", 0.8934603273216726}]
> Running step 48cc5958-fb9c-4257-8b39-7db5b8bb6785. Step input: None
Thought: The recommended books do not seem to match the user's request for a book containing decorative and traditional patterns and illustrations from regions nearby Iran. I need to try another approach.
Action: getBookDescription
Action Input: {'book': 'A pictorial history of magic and the supernatural'}
Observation: A collection of illustrations and photographs describing magic and supernatural occurrences throughout history.
> Running step e222c1cf-0d80-4d2c-a815-2c8ede120692. Step input: None
Thought: The book "A pictorial history of magic and the supernatural" does not match the user's request as it is about magic and supernatural occurrences, not decorative and traditional patterns and illustrations from regions nearby Iran. I should try to recommend books based on the genre of the book that the user is looking for.
Action: recommendSameGenreAs
Action Input: {'book_title': 'Persian Miniatures', 'top_k': 5}
Observation: []
> Running step 7699e7c4-c4e8-4d70-838a-b7d14d2b53b3. Step input: None
Thought: The tool did not return any recommendations for books with a similar genre to "Persian Miniatures". This could be because the book is not in the database or there are no similar books. I'll try to find books by authors who specialize in decorative and traditional patterns and illustrations from regions nearby Iran.
Action: getBooksFromAuthor
Action Input: {'author': 'authors who specialize in decorative and traditional patterns and illustrations from regions nearby Iran'}
Observation: []
> Running step 99d4a979-a8b9-4823-9b05-0d66bd4a7ded. Step input: None
Thought: The tool did not return any books written by authors who specialize in decorative and traditional patterns and illustrations from regions nearby Iran. This could be because there are no such authors or books in the database, or because the query was too specific. I should try to find a different approach.
Action: recommendBooksByReviews
Observation: [{"Noble House", 0.872492475915324}]
> Running step c407a1b3-bef3-4bf1-ab78-98fb11024762. Step input: None
Thought: The recommended books do not seem to match the user's request for a book containing decorative and traditional patterns and illustrations from regions nearby Iran. I've tried various approaches, but none of them have yielded relevant results. At this point, I must conclude that I cannot find a suitable book using the available tools.
Answer: Unfortunately, I couldn't find any books that match your request for decorative and traditional patterns and illustrations from regions nearby Iran. You might want to try searching online or visiting a library to see if they have any books that fit your criteria.
```

Figura 4.1.9 - Logs al solicitar recomendación de libro en inglés. Elaboración propia

Llama la atención de los logs cómo el agente ha utilizado varias herramientas. Entre ellas, la recomendación de libros y la recomendación a partir de las reseñas. Esto puede deberse a un **intento del agente de contrastar información** al disponer de dos herramientas muy similares. Por otra parte, ha utilizado herramientas que no han proporcionado resultados, y aun así, ha razonado y sabido continuar para responder al usuario.

## 4.2 Análisis D.A.F.O.

En los siguientes subapartados se realizará un análisis del sistema, contemplando las debilidades, amenazas, fortalezas y oportunidades del mismo.

### 4.2.1 Debilidades

Entre las debilidades del sistema destacan

- Rigidez de Streamlit. Como se ha expresado previamente, es un *framework* muy sencillo que no añade complejidad al código, pero tampoco otorga flexibilidad al desarrollador. El mayor ejemplo de esto es la imposibilidad de poder posicionar el botón del micrófono en la zona de entrada de texto del usuario.
- BBDD no construida de forma óptima. Pese a que se tiene una gran cantidad de datos, probablemente se pueda y deba aprovechar más la BBDD orientada a grafos creando más nodos con menos propiedades para centrarse en las relaciones.
- El sistema no mantiene un registro de los usuarios ni de sus gustos. Hace las recomendaciones de manera estática conforme a lo almacenado en la base de datos, la cual no se actualiza de forma automática.

- Al usar la transcripción, se utiliza la entrada de audio del sistema, no del navegador. Esto impide que la aplicación se ejecute a través de la red o como sistema distribuido.
- El sistema de transcripción no es instantáneo, ya que el modelo debe cargarse en memoria antes de poderse utilizar.
- Los *embeddings* de la BBDD hacen que la misma crezca de tamaño de forma importante, impidiendo la ejecución de la aplicación en múltiples dispositivos.

#### 4.2.2 Amenazas

Las principales amenazas son:

- Lentitud. Al ejecutarse todo en local, depende de los recursos del sistema, que suelen ser menores que los disponibles en la nube. Por eso el tiempo para generar las respuestas es mucho mayor que si se ejecutara en la nube.
- Existen soluciones similares disponibles online que realizan el mismo servicio de forma actualizada y teniendo en cuenta las opiniones de los usuarios.
- Aparición de nuevos modelos más avanzados que no se puedan ejecutar en local, por lo que no son aplicables al sistema.
- El sistema de transcripción no muestra en tiempo real las palabras que va reconociendo, mientras que existen otros sistemas que sí lo hacen.

#### 4.2.3 Fortalezas

- Este sistema no necesita conexión a Internet para funcionar. Además, permite mantener la gobernanza de los datos al no necesitar enviarlos a terceros.
- Todos los datos que haya en la BBDD serán totalmente privados ya que no se envían a ningún servidor externo, ni de LLMs.
- Se evitan alucinaciones gracias a las herramientas RAG de las que dispone el agente.
- Permite el acceso a personas con movilidad reducida dado que admite una entrada por voz para interactuar con el *recomendador*.
- La BBDD generada y utilizada es lo suficientemente grande como para tomarla de base en un caso de uso real. Además se podría seccionar para segmentarla para diferentes públicos, como entretenimiento, educacional, etc.



#### 4.2.4 Oportunidades

- Al tener desacoplada la interfaz de la lógica y de los datos del sistema, el *frontend* puede ser sustituido con facilidad, solucionando los problemas presentados por Streamlit.
- Dado que se dispone de logs del agente, se puede añadir a la interfaz de usuario los pensamientos y acciones según se van desarrollando para que el usuario vea cómo surge la respuesta a su consulta.
- Se podría utilizar un sistema de login para tener control de los usuarios y, dado que la BBDD es orientada a grafos, se podrían añadir los gustos y preferencias de los usuarios para mejorar el sistema con el tiempo y uso.
- Dado que la BBDD contiene enlaces a imágenes de los libros, se podrían utilizar para mostrar junto con las recomendaciones a través de la interfaz de usuario. Haría la aplicación mucho más intuitiva y directa.

#### 4.3 Líneas de futuro

A partir de los resultados obtenidos y el análisis DAFO, se identifican diversas áreas de mejora y expansión para el sistema desarrollado. Estas líneas de futuro tienen como objetivo incrementar la funcionalidad, escalabilidad y usabilidad del *recomendador*, a la vez que explorar nuevas oportunidades:

- **Integración de perfiles de usuario:** Incorporar un sistema de registro y gestión de perfiles de usuario que permita almacenar sus preferencias, historial de búsquedas y recomendaciones personalizadas. Esto facilitará ofrecer recomendaciones dinámicas y ajustadas a los intereses individuales, mejorando la experiencia del usuario.
- **Ampliación de la BBDD:** Explorar y añadir *datasets* complementarios, como metadatos más detallados de los libros (por ejemplo, premios recibidos o críticas destacadas), y géneros no contemplados en la base actual. Esto enriquecerá el alcance y la precisión de las recomendaciones.
- **Optimización de la BBDD:** Rediseñar el modelo de la base de datos para mejorar la representación de relaciones complejas entre usuarios, autores, géneros y reseñas. Esto podría incluir el uso de índices específicos y nodos adicionales para consultas más rápidas y eficientes.
- **Incorporación de análisis de sentimientos:** Implementar modelos de análisis de sentimientos para evaluar las opiniones de las reseñas, proporcionando recomendaciones basadas no solo en similitudes de contenido, sino también en las emociones y valoraciones expresadas por los usuarios.
- **Implementación de nuevas interfaces de usuario:** Desarrollar una interfaz más flexible, posiblemente basada en *frameworks* web avanzados como React o Angular, que permita una experiencia más interactiva y adaptable a



diferentes dispositivos. Esto resolvería las limitaciones detectadas en el uso de Streamlit.

- **Expansión hacia un sistema distribuido:** Migrar la infraestructura del *recomendador* a un entorno basado en la nube para aprovechar recursos computacionales escalables, permitiendo un acceso más rápido y eficiente desde cualquier dispositivo y eliminando las limitaciones de procesamiento local. Esto incluiría usar LLMs en la propia nube, o usar las APIs de terceros si la privacidad no es esencial—ya que económicamente suele ser rentable.
- **Integración de tecnologías emergentes:** Explorar el uso de modelos más avanzados de LLMs o *embeddings* disponibles en el futuro, asegurando compatibilidad con APIs externas para mejorar la calidad y versatilidad del sistema. Asimismo, permitir el uso del sistema a través de una API para que otros desarrolladores puedan construir aplicaciones que utilicen como base este sistema *recomendador*.
- **Soporte para consultas multimedia:** Ampliar las capacidades del sistema para admitir búsquedas y recomendaciones basadas en contenido visual o auditivo, como portadas de libros o audiolibros, integrando tecnologías como reconocimiento de imágenes y procesamiento de audio. Generar las respuestas mediante un vídeo en el que un humano generado por IA responda al usuario.
- **Gamificación y recomendaciones colaborativas:** Introducir elementos de gamificación, como logros y retos, que incentiven el uso del sistema. Además, incorporar algoritmos de filtrado colaborativo que aprovechen datos de otros usuarios para mejorar las recomendaciones. Útil para incentivar el uso del sistema entre los más jóvenes.
- **Monetización y aplicaciones comerciales:** Analizar oportunidades de monetización, como integraciones con plataformas de comercio electrónico o modelos de suscripción premium que ofrezcan funcionalidades adicionales, como recomendaciones más precisas o análisis avanzados.

## 4.4 Conclusiones

El desarrollo de este proyecto ha permitido explorar en profundidad las capacidades de los sistemas *recomendadores*, demostrando cómo es posible **combinar diversas tecnologías para abordar un problema real**, como es la personalización de recomendaciones de libros. A lo largo del proceso, se integraron **herramientas avanzadas**, desde bases de datos orientadas a grafos hasta modelos de lenguaje de gran tamaño (LLMs), y se **superaron múltiples retos** relacionados con la implementación y optimización de un sistema funcional.

Una de las principales conclusiones es que, aunque las tecnologías actuales ofrecen un gran potencial para la creación de sistemas inteligentes, **la calidad de los datos sigue siendo un factor crítico**. El uso del *dataset* “Amazon Book Reviews” permitió construir una base sólida para el sistema, pero también evidenció la necesidad de preprocesar y limpiar los datos para evitar errores en las etapas posteriores del desarrollo. Además, el **almacenamiento de *embeddings* en la base de datos** fue clave para garantizar la eficiencia en la búsqueda y recomendación de contenido, a la vez que un problema al aumentar exponencialmente el tamaño de la misma.

El proyecto también destacó la importancia de los **agentes ReACT y las herramientas RAG como elementos fundamentales para reducir alucinaciones** de los LLMs y mejorar la precisión de las respuestas generadas. Estos componentes demostraron ser esenciales para proporcionar recomendaciones basadas en datos reales y no únicamente en el conocimiento entrenado del modelo de lenguaje.

En cuanto al diseño y la implementación, el uso de *frameworks* como **Streamlit permitió desarrollar rápidamente una interfaz** funcional y visualmente atractiva, aunque sus limitaciones en términos de personalización pusieron de manifiesto la necesidad de explorar alternativas más flexibles en futuros desarrollos. Asimismo, la integración de un sistema de transcripción de voz con **Whisper añadió una capa de accesibilidad** y modernidad al sistema, aunque evidenció retos asociados con el uso de modelos en tiempo real en dispositivos con recursos limitados.

Finalmente, este proyecto ha sido un ejemplo claro de cómo los sistemas de información y la inteligencia artificial pueden converger para crear soluciones innovadoras y personalizadas. No solo se logró cumplir con los objetivos planteados, sino que se abrieron múltiples líneas de investigación y desarrollo futuro que podrían ampliar significativamente el alcance y la aplicabilidad del sistema. Este trabajo también ha servido como una plataforma de aprendizaje, permitiendo profundizar en tecnologías clave como bases de datos orientadas a grafos, generación de *embeddings*, y agentes autónomos basados en LLMs, estableciendo una base sólida para proyectos más complejos en el futuro.

Además, gracias al trabajo desarrollado y al tiempo dispuesto para ello a lo largo de esta asignatura, he sido capaz de aprender varias tecnologías, como ya se ha mencionado previamente. De ellas destaco los cursos que fui realizando de Neo4j, para los cuales adjunto los certificados obtenidos:

- **Neo4j Fundamentals:** fundamentos de Neo4j y el modelo de grafo de propiedades. [26]
- **Cypher Fundamentals:** aprender Cypher en 1 hora. [27]
- **Introduction to Neo4j Graph Data Science:** obtención de una comprensión técnica de alto nivel de la biblioteca Graph Data Science (GDS). [28]

- **Introduction to Neo4j & GraphQL:** Aprender cómo crear GraphQL APIs usando Neo4j GraphQL Toolbox y biblioteca. [29]
- **Graph Data Modeling Fundamentals:** aprender cómo diseñar un grafo Neo4j siguiendo las mejores prácticas. [30]
- **Neo4j Graph Data Science Fundamentals:** aprender todo lo necesario sobre algoritmos de grafos y procesos de aprendizaje automático. [31]
- **Building Neo4j Applications with Python:** aprender cómo interactuar con Neo4j desde Python usando Neo4j Python Driver. [32]
- **Importing CSV data into Neo4j:** aprender cómo importar datos de CSV en Neo4j usando Cypher. [33]
- **Introduction to Vector Indexes and Unstructured Data:** entender y buscar datos no estructurados usando índices vectoriales. [34]
- **Building Knowledge Graphs with LLMs:** aprender cómo usar IA generativa, LLMs y Python para convertir datos no estructurados en grafos. [35]
- **Cypher Indexes and Constraints:** mejorar el rendimiento de tu grafo con restricciones e índices de Cypher. [36]
- **Importing Data Fundamentals:** aprender cómo importar datos en Neo4j. [37]
- **Neo4j & LLM Fundamentals:** aprender cómo usar Neo4j con LLMs. [38]
- **Build a Neo4j-backed Chatbot using Python:** construye un chatbot usando Neo4j, LangChain y Streamlit. [39]
- **Intermediate Cypher Queries:** continúa tu viaje de aprendizaje con las consultas de Cypher. [40]

Finalmente, cabe destacar que obtuve la certificación **Neo4j Certified Professional**. [41]

## 5 Bibliografía

- [1] MuleSoft, Salesforce Inc., «¿Qué es una API? (Interfaz de programación de aplicaciones) | MuleSoft,» [En línea]. Available: <https://www.mulesoft.com/es/resources/api/what-is-an-api>. [Último acceso: 1 Enero 2025].
- [2] IBM, «¿Qué es un chatbot? | IBM,» [En línea]. Available: <https://www.ibm.com/es-es/topics/chatbots>. [Último acceso: 1 Enero 2025].
- [3] Wikipedia, «GitHub - Wikipedia, la enciclopedia libre,» [En línea]. Available: <https://es.wikipedia.org/wiki/GitHub>. [Último acceso: 1 Enero 2025].
- [4] Hugging Face Inc., «Hugging Face – The AI community building the future.,» [En línea]. Available: <https://huggingface.co/>. [Último acceso: 1 Enero 2025].
- [5] Jupyter, «Project Jupyter | Home,» [En línea]. Available: <https://jupyter.org/>. [Último acceso: 5 Enero 2025].
- [6] Wikipedia, «Kaggle - Wikipedia, la enciclopedia libre,» [En línea]. Available: <https://es.wikipedia.org/wiki/Kaggle>. [Último acceso: 1 Enero 2025].
- [7] LlamaIndex, «LlamaIndex - LlamaIndex,» [En línea]. Available: <https://docs.llamaindex.ai/en/stable/>. [Último acceso: 5 Enero 2025].
- [8] Amazon Web Services, «¿Qué es un LLM (modelo de lenguaje de gran tamaño)?,» [En línea]. Available: <https://aws.amazon.com/es/what-is/large-language-model/>. [Último acceso: 1 Enero 2025].
- [9] Leader GPU, «¿Cómo funciona Ollama?,» [En línea]. Available: <https://www.leadergpu.es/articles/534-c-mo-funciona-ollama#:~:text=Integraci%C3%B3n%20de%20VSCode-,Qu%C3%A9%20es%20Ollama,de%20datos%20confidenciales%20y%20sensibles..> [Último acceso: 1 Enero 2025].
- [10] Python Software Foundation, «Welcome to Python.org,» [En línea]. Available: <https://www.python.org/>. [Último acceso: 1 Enero 2025].
- [11] Amazon Web Services, «¿Qué es la RAG (generación aumentada por recuperación)?,» [En línea]. Available: <https://aws.amazon.com/es/what-is/retrieval-augmented-generation/>. [Último acceso: 1 Enero 2025].

- [12] Spotify, «Centro de seguridad y privacidad | Spotify,» [En línea]. Available: <https://www.spotify.com/es/safetyandprivacy/understanding-recommendations>. [Último acceso: 1 Enero 2025].
- [13] Snowflake Inc., «Streamlit • A faster way to build and share data apps,» [En línea]. Available: <https://streamlit.io/>. [Último acceso: 1 Enero 2025].
- [14] TensorFlow, «Citando a TensorFlow,» [En línea]. Available: <https://www.tensorflow.org/about/bib?hl=es-419>. [Último acceso: 5 Enero 2025].
- [15] Wikipedia, «Unidad de procesamiento tensorial - Wikipedia, la enciclopedia libre,» [En línea]. Available: [https://es.wikipedia.org/wiki/Unidad\\_de\\_procesamiento\\_tensorial](https://es.wikipedia.org/wiki/Unidad_de_procesamiento_tensorial). [Último acceso: 5 Enero 2025].
- [16] Wikipedia, «Word embedding - Wikipedia, la enciclopedia libre,» [En línea]. Available: [https://es.wikipedia.org/wiki/Word\\_embedding](https://es.wikipedia.org/wiki/Word_embedding). [Último acceso: 1 Enero 2025].
- [17] Amazon Web Services, «¿Qué son los agentes de inteligencia artificial?: explicación de los agentes de inteligencia artificial - AWS,» [En línea]. Available: <https://aws.amazon.com/es/what-is/ai-agents/>. [Último acceso: 5 Enero 2025].
- [18] Creative Commons, «Deed - CC0 1.0 Universal - Creative Commons,» [En línea]. Available: <https://creativecommons.org/publicdomain/zero/1.0/>. [Último acceso: 1 Enero 2025].
- [19] Kaggle, «Amazon Books Reviews | Kaggle,» [En línea]. Available: <https://www.kaggle.com/datasets/mohamedbakhmet/amazon-books-reviews>. [Último acceso: 1 Enero 2025].
- [20] Neo4j, «Awesome Procedures On Cypher (APOC) - Neo4j Labs,» [En línea]. Available: <https://neo4j.com/labs/apoc/>. [Último acceso: 3 Enero 2025].
- [21] Hugging Face, «Models - Hugging Face,» [En línea]. Available: <https://huggingface.co/models?other=text-embedding>. [Último acceso: 4 Enero 2025].
- [22] Hugging Face, «MTEB Leaderboard - a Hugging Face Space by mteb,» [En línea]. Available: <https://huggingface.co/spaces/mteb/leaderboard>. [Último acceso: 4 Enero 2025].
- [23] Hugging Face, «dunzhang/stella\_en\_1.5B\_v5 · Hugging Face,» [En línea]. Available: [https://huggingface.co/dunzhang/stella\\_en\\_1.5B\\_v5](https://huggingface.co/dunzhang/stella_en_1.5B_v5). [Último acceso: 4 Enero 2025].

- [24] Google, «colab.google,» [En línea]. Available: <https://colab.google/>. [Último acceso: 5 Enero 2025].
- [25] davabase, «davabase/whisper\_real\_time: Real time transcription with OpenAI Whisper.,» [En línea]. Available: [https://github.com/davabase/whisper\\_real\\_time](https://github.com/davabase/whisper_real_time). [Último acceso: 6 Enero 2025].
- [26] Neo4j, «Neo4j Fundamentals | Álvaro Prieto Álvarez's Achievements,» 17 Octubre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/02c6e85d-50cc-4c6a-a415-c2464baf8f3e/>. [Último acceso: 1 Enero 2025].
- [27] Neo4j, «Cypher Fundamentals | Álvaro Prieto Álvarez's Achievements,» 17 Octubre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/94bd7ed2-9bad-40d5-b980-19c2321cfb5f/>. [Último acceso: 1 Enero 2025].
- [28] Neo4j, «Introduction to Neo4j Graph Data Science | Álvaro Prieto Álvarez's Achievements,» 31 Octubre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/7626ccf8-0c96-432d-b11b-0e37b59fddc3/>. [Último acceso: 1 Enero 2025].
- [29] Neo4j, «Introduction to Neo4j & GraphQL | Álvaro Prieto Álvarez's Achievements,» 6 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/754f1399-7a96-415c-aadc-8f41ac9e4239/>. [Último acceso: 1 Enero 2025].
- [30] Neo4j, «Graph Data Modeling Fundamentals | Álvaro Prieto Álvarez's Achievements,» 6 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/a128555c-406d-4983-a40a-ad0212051fb5/>. [Último acceso: 1 Enero 2025].
- [31] Neo4j, «Neo4j Graph Data Science Fundamentals | Álvaro Prieto Álvarez's Achievements,» 5 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/0fb3d802-ba00-4fae-bda5-751abdf1f707/>. [Último acceso: 1 Enero 2025].
- [32] Neo4j, «Building Neo4j Applications with Python | Álvaro Prieto Álvarez's Achievements,» 11 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/29aa4f3b-2ce2-4ad1-b2e4-1fc51789144e/>. [Último acceso: 1 Enero 2025].
- [33] Neo4j, «Importing CSV data into Neo4j | Álvaro Prieto Álvarez's Achievements,» 13 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/ca8f0c92-7b97-4c73-956b-fc9d3c57fbef/>. [Último acceso: 1 Enero 2025].

- [34] Neo4j, «Introduction to Vector Indexes and Unstructured Data | Álvaro Prieto Álvarez's Achievements,» 24 Septiembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/48e2606e-47d3-46e7-800b-04e4c43b6e63/>. [Último acceso: 1 Enero 2025].
- [35] Neo4j, «Building Knowledge Graphs with LLMs | Álvaro Prieto Álvarez's Achievements,» 18 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/663fa74a-c649-45a3-b8f5-8ed6cdb3dd6a/>. [Último acceso: 1 Enero 2025].
- [36] Neo4j, «Cypher Indexes and Constraints | Álvaro Prieto Álvarez's Achievements | Free Neo4j Courses from GraphAcademy,» 18 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/e48246d7-1460-459e-9c39-4a1867125785/>. [Último acceso: 1 Enero 2025].
- [37] Neo4j, «Importing Data Fundamentals | Álvaro Prieto Álvarez's Achievements | Free Neo4j Courses from GraphAcademy,» 20 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/9a2ccae9-c10d-4975-b167-592d2720bc16/>. [Último acceso: 1 Enero 2025].
- [38] Neo4j, «Neo4j & LLM Fundamentals | Álvaro Prieto Álvarez's Achievements | Free Neo4j Courses from GraphAcademy,» 6 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/28118a36-d897-44da-aa65-9764f67773c8/>. [Último acceso: 1 Enero 2025].
- [39] Neo4j, «Build a Neo4j-backed Chatbot using Python | Álvaro Prieto Álvarez's Achievements | Free Neo4j Courses from GraphAcademy,» 25 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/5823b631-85b2-477f-945e-ef5612852b52/>. [Último acceso: 1 Enero 2025].
- [40] Neo4j, «Intermediate Cypher Queries | Álvaro Prieto Álvarez's Achievements | Free Neo4j Courses from GraphAcademy,» 2 Diciembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/8a65a035-0d72-463e-a7f3-4eb26fa604fe/>. [Último acceso: 1 Enero 2025].
- [41] Neo4j, «Neo4j Certified Professional | Álvaro Prieto Álvarez's Achievements | Free Neo4j Courses from GraphAcademy,» 20 Noviembre 2024. [En línea]. Available: <https://graphacademy.neo4j.com/c/88146a2a-894e-4614-bac0-cef1a1338b39/>. [Último acceso: 1 Enero 2025].
- [42] MuleSoft, Salesforce Inc., «¿Qué es una API? (Interfaz de programación de aplicaciones),» [En línea]. Available:

<https://www.mulesoft.com/es/resources/api/what-is-an-api>. [Último acceso: 02 Diciembre 2024].