universidad de león

Escuela de Ingenierías
Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Sistema de recomendación de portátiles basado en LLMs.

Autor: Jorge Alonso Fernández.

Índice.

1.Introducción.

El objetivo del proyecto consistirá en la creación de un sistema RAG cuya función sea ser un recomendador de un ámbito concreto.

El campo de nuestro sistema será el mercado de los ordenadores portátiles, el sistema ayudará a los usuarios a encontrar el equipo que más se adapte a sus necesidades facilitándoles así la decisión de compra.

2. Explicación del problema.

Como informático, es muy probable que personas de tu entorno, con pocos conocimientos sobre componentes electrónicos, te pidan recomendaciones para poder realizar compras eficientes.
La idea de este sistema es dar solución al problema de recomendar ordenadores portátiles a usuarios con distintos niveles de conocimiento, desde personas con nociones muy básicas hasta usuarios más avanzados.

3.Elección de los Datos.

3.1. Elección del Dataset.

Antes de iniciar con la construcción de la base de datos relacional, necesitamos obtener un conjunto de datos que sirva de base para su creación y que, a su vez, nuestro modelo sea capaz de interpretar y procesar dicha información de forma adecuada.

Ya que nuestro modelo se encargará de recomendar portátiles a los usuarios, debemos de tener un Dataset que contenga información sobre sus características, para poder analizarlo y llevar a cabo la recomendación.

El Dataset elegido ha sido obtenido de Kaggle la cual es una plataforma online y una comunidad global de científicos de datos y expertos en aprendizaje automático.

El Dataset elegido se llama “Laptop Price and Specification Dataset” (https://www.kaggle.com/datasets/sumanbera19/laptop-price-dataset).

3.2 Modificación del Dataset.

El Dataset cuenta con las siguientes características para cada ordenador: Model, Price, Rating, Generation, Core, Ram, SSD, Display, Graphics, OS y Waranty.

En nuestro caso no necesitaremos todas las características anteriores, nos quedaremos solo con las necesarias, a continuación, mostramos las características tomadas del Dataset anterior junto con una descripción de estas:

- Model: Nombre o modelo del portátil. Incluye la marca y la serie o nombre del producto (p. ej., “HP Pavilion x360”, “Dell Inspiron 15”).
- Price: Precio de venta del portátil, probablemente en rupias indias (INR). Es un valor numérico y puede usarse como variable objetivo en tareas de regresión (predicción de precio).
- Generation: Generación del procesador del portátil (p. ej., 10ª Gen, 11ª Gen). Ayuda a identificar la clase de rendimiento de la CPU.
- Core: Serie del procesador (p. ej., Intel i3, i5, i7; AMD Ryzen 3, Ryzen 5, etc.). Factor clave de la potencia de procesamiento.
- Ram: Cantidad de memoria del sistema (p. ej., 8 GB, 16 GB). Más RAM suele implicar mejor rendimiento en multitarea.
- SSD: Capacidad del almacenamiento de estado sólido (p. ej., 256 GB, 512 GB). Si falta, puede que use HDD o no se haya dado esa información.
- Display: Tamaño y descripción de resolución (p. ej., 15,6" FHD, 14" HD). Afecta a la calidad de visualización y portabilidad.
- Graphics: Tipo de tarjeta gráfica. Puede ser integrada (p. ej., Intel UHD Graphics) o dedicada (p. ej., NVIDIA GeForce). Relevante para juegos o trabajo gráfico exigente.

Finalmente, se ha añadido al Dataset la característica “type”, cuyo objetico es agrupar los portátiles en diferentes categorías con el fin de simplificar y mejorar los procesos de búsqueda. A continuación, mostramos las categorías definidas junto con una descripción de estas:

- Juegos (Gaming): Diseñados para alto rendimiento en videojuegos y tareas gráficas exigentes.
- Oficina: Enfocados en productividad, trabajos administrativos y uso diario.
- Ultraportátil: Ligeros, compactos y con gran autonomía, pensados para movilidad.
- General: Modelos versátiles que equilibran rendimiento, precio y funcionalidad.
- Estación de trabajo (Workstation): Portátiles profesionales de gama alta para tareas técnicas o de ingeniería.

Por lo que las características finales de nuestro Dataset serán las siguientes: Model, Price, Generation, Core, Ram, SSD, Display, Graphics y type.

4. Construcción de la Base de Datos.

4.1. Definir la Estructura de la Base de Datos.

Para construir la base de datos usaremos Neo4j, el cual es una base de datos de grafos que organiza la información como nodos (entidades) y relaciones (vínculos entre ellas).

Para nuestra base de datos de portátiles hemos definido los siguientes nodos y relaciones:

Nodos:

Las propiedades de los Nodos han sido obtenidas a partir de las diferentes características del Dataset.

- Company: representa la marca o fabricante su propiedad principal es “name”, esta propiedad la extraemos mediante una función de la característica del Dataset llamada “model”
- Laptop: representa cada modelo de portátil sus propiedades son:
    - id: compuesto a partir de **“**Company” **+ “**Model” (limpiando espacios).
    - product: **“**Model” (texto tal cual del dataset).
    - cpu: **“**Generation” (cadena con la generación de la CPU).
    - gpu: **“**Graphics”**.**
    - ram: **“**Ram” (parseado a número de GB).
    - inches: **“**Display” (parseado a pulgadas; si incluye resolución, se descarta para este campo).
    - price: **“**Price” (normalizado a EUR desde INR).
    - memory: “SSD”
    - url: obtenido mediante Scraping (ver apartado 3.2).
- Type: representa la categoría del portátil (p. ej., Gaming, Oficina, Ultraportátil, General, Workstation; propiedad principal: “name” obtenida de la característica del Dataset llamada “type”).

Relaciones:

Company → FABRICA → Laptop: una marca fabrica uno o varios portátiles.

Laptop → ES_TIPO_DE → Type: cada portátil pertenece a una categoría.

4.2. Uso de Scraping.

Para la obtención de las propiedades de los nodos usamos el Dataset obtenido y modificado, tal y como se mencionó en los apartados anteriores.

Sin embargo, quise añadir la propiedad “url” para que los usuarios pudieran ver de forma más detallada y visual los portátiles. Dado que el Dataset no contenía esta información, y como se comentó en clase, decidí utilizar la herramienta Selenium para “scrapear”.

Gracias a Selenium, puedo obtener enlaces que llevan a páginas de compra de los ordenadores y mostrárselos a los usuarios. Cuando se crea cada nodo Laptop, gracias al nombre del portátil y a un

script que usa Selenium, se obtienen los links y se guardan en la base de datos como propiedad del nodo.

4.3 Embeddings e índice vectorial.

Para que el recomendador entienda mejor lo que pide el usuario, a cada portátil le generamos un embedding, que es una representación numérica basada en una descripción textual del propio equipo. Esta información se construye usando la información del dataset y unas etiquetas semánticas que definiremos nosotros y explicaremos a continuación la razón de estas etiquetas.

El problema de nuestros datos es que si generamos embeddings de datos sueltos (por ejemplo, “16 GB”, “i5”, “15.6”) nuestra semántica seria muy pobre, el modelo identificaría si son números o cadenas, pero no entendería su uso real. Para solucionar este problema se analizan las propiedades de forma individual construyendo una descripción bien estructurada en la que cada atributo se le asignan una serie de palabras clave y frases cortas que logran dotar contexto para que el sistema pueda comprenderlo.

A partir de cada fila del dataset generamos un texto que incluye:

- Datos básicos del portátil:
    - model: nombre completo del modelo.
    - brand: marca extraída del modelo.
    - type: categoría del portátil (juegos, oficina, general, ultraportátil, estación de trabajo).
    - cpu_full: descripción completa de la CPU.
    - gpu_full: descripción completa de la GPU.
    - ram: cantidad de memoria RAM en GB.
    - screen_inches: tamaño de pantalla en pulgadas.
    - storage: tipo y capacidad de almacenamiento (por ejemplo, “512GB SSD”).
    - price_eur: precio en euros.
- Atributos derivados y normalizados, que ayudan a comparar portátiles entre sí:
    - cpu_simple: versión simplificada de la CPU (i3, i5, i7, i9, ryzen3, ryzen5, etc.).
    - gpu_number: modelo numérico de la GPU cuando es reconocible (3050, 4060, 4090…).
    - gpu_type: si la gráfica es integrada o dedicada.
    - price_tier: tramo de precio (bajo, medio o alto) en función del valor en euros.

- Etiquetas semánticas en lenguaje natural, que describen para qué es adecuado el equipo según su hardware y su tipo. Por ejemplo:
    - Para portátiles de juegos o con GPU dedicada potente: “gaming, videojuegos, alto rendimiento gráfico, FPS altos”.
    - Para portátiles de oficina: “ofimática, trabajo de oficina, Excel, Word, correo, navegación web”.
    - Para portátiles pensados para estudiar: “uso general, estudiantes, clases, universidad, videollamadas”.
    - Para ultraportátiles: “portátil ligero, fácil de transportar, movilidad, buena batería”.
    - Para estaciones de trabajo o hardware muy potente: “edición de vídeo, modelado 3D, renderizado, datos pesados, inteligencia artificial”.
    - Según la RAM, el tamaño de pantalla y el precio se añaden también etiquetas como “buena multitarea”, “pantalla grande para multimedia”, “presupuesto ajustado” o “alta gama”.

El resultado es una cadena de texto similar a:

model: HP Pavilion 15 | brand: HP | type: general | cpu_full: intel core i5 | cpu_simple: i5 | gpu_full: iris xe graphics | gpu_type: integrated | ram:16GB | screen_inches:15.6 | storage:512GB SSD | price_eur:699 | price_tier:mid | semantic: uso general hogar estudiantes clases universidad ofimática…

A la cadena final obtenida se le aplica el modelo de embedding, y se convierte en un vector numérico que gracias a las etiquetas añadidas estará dotado de suficiente contexto para la comprensión correcta y eficiente del sistema.

4.4. Construcción de la Base de Datos.

Para la construcción de la Base de datos usamos el script database_generator.py, que automatiza todo el proceso y sigue las siguientes fases:

1. Punto de partida: Usamos el CSV que seleccionamos en 2.1 y limpiamos en 2.2 (con los campos finales y la columna *type*).
2. Lanzamos el programa: Lee el CSV fila a fila y prepara los datos: deja la marca y el modelo bien separados, la RAM en GB, la pantalla en pulgadas y el precio convertido a euros.
3. Añade el enlace: Tal y como explicamos en 3.2, hace scraping para intentar conseguir una URL de compra del portátil y la guarda si la encuentra.
4. Guarda en Neo4j: Crea la marca (Company), el portátil (Laptop) y su categoría (Type) y los deja conectados entre sí para que luego sea fácil consultar por marca o por tipo.

5. Activa la búsqueda por significado: para cada portátil genera el texto de descripción semántica, obtiene el embedding correspondiente y lo guarda como propiedad embedding del nodo Laptop.
6. Crea el índice vectorial: una vez insertados los portátiles, se crea en Neo4j un índice vectorial sobre el campo embedding (por ejemplo, laptop_embeddings), que permite recuperar los modelos más cercanos a una consulta en función de la similitud de sus vectores.

Finalmente, queda una base de datos ordenada, con enlaces y categorías, lista para que el sistema recomiende portátiles según lo que pida el usuario.

5. Arquitectura del sistema RAG.

5.1. Interfaz (Streamlit).

Streamlit es una biblioteca de Python de código abierto que simplifica la creación de aplicaciones web interactivas para ciencia de datos y aprendizaje automático. Permite a los desarrolladores y científicos de datos construir interfaces de usuario rápidamente usando solo Python

La interfaz es la forma en la que el usuario interactúa con el sistema. En nuestra implementación presenta un chat limpio, conserva el historial y muestra las respuestas con especificaciones clave (CPU, GPU, RAM, pantalla, precio y enlace cuando existe).

5.2 LlamaIndex (orquestador).

LlamaIndex es un marco de código abierto para orquestar datos que simplifica la conexión de modelos de lenguaje grandes (LLMs) con fuentes de datos privadas, como documentos, bases de datos y APIs.

LlamaIndex funciona como un “coordinador” entre las diferentes partes del sistema. En nuestro sistema, tras recibir la pregunta desde la interfaz:

- Primero aplica una capa de moderación y limpieza a través del LLM (Groq), para asegurarse de que la consulta trata realmente sobre portátiles y que el texto es apto;
- Después extrae filtros básicos de la pregunta (por ejemplo, precio máximo, RAM mínima, GPU concreta, pulgadas o tipo de uso);
- En función de si la consulta contiene especificaciones técnicas claras, decide si debe hacer una búsqueda simbólica en Neo4j (por filtros exactos) o una búsqueda semántica basada en embeddings;

- Finalmente combina los resultados y prepara la información para que el LLM pueda generar una respuesta explicada.

5.3 Ollama (Construcción de Embeddings).

Ollama es un entorno de ejecución local para modelos de lenguaje y modelos de embeddings que permite generar representaciones vectoriales sin depender de servicios externos ni de computación en la nube.

Ollama se encarga de transformar descripciones de portátiles y preguntas en vectores comparables. Se calculan de forma local y alimentan tanto la construcción de la base de datos (embeddings de cada Laptop) como la búsqueda semántica de las consultas del usuario.

5.4 Neo4j (Base de datos).

Neo4j es una base de datos orientada a grafos que almacena datos en nodos y sus relaciones en aristas, lo que permite gestionar de forma eficiente las conexiones entre los datos.

Neo4j se usa para la construcción de una Base de datos basada en Grafos con las entidades mencionadas en el apartado 3.1 enriquecidas con información de precio, características técnicas, URL y embedding vectorial. Sobre estos embeddings se construye un índice vectorial, que permite recuperar los portátiles más parecidos a la consulta.

5.5 Groq (LLM).

Groq es una plataforma de inferencia de modelos de lenguaje diseñada para ofrecer ejecuciones extremadamente rápidas y eficientes mediante hardware especializado y una arquitectura optimizada.

Groq nos proporciona el modelo de lenguaje y la capa de razonamiento del sistema. Sus funciones principales son:

- Interpretar la consulta del usuario y determinar si es tratable y está relacionada con portátiles;
- Ayudar a completar lagunas (por ejemplo, asumir un uso esperado cuando el usuario no lo especifica);
- Aplicar condiciones duras (precio máximo, RAM mínima, uso previsto, etc.) a la lista de candidatos;

- Redactar una respuesta breve y ordenada justificando por qué encaja cada modelo recomendado.

5.6. ¿Por qué Groq?

Considero importante este punto para explicar por qué se tomó la decisión de usar Groq. En un principio, cuando el programa fue creado, se usaba como LLM el modelo local de Ollama (mistral:7b), sin embargo, esto causaba problemas de memoria que no permitían el funcionamiento del programa y las respuestas tardaban mucho en procesarse. Por eso se tomó la decisión de usar Groq para corregir estas desventajas.

5.7. Funcionamiento del sistema en conjunto

Tras explicar las tecnologías que se han usado para construir nuestro sistema RAG, ahora nos centraremos en explicar cómo funciona el sistema cuando un usuario hace una consulta y se genera una recomendación.

Para ello, describimos el flujo completo de principio a fin:

1. **El usuario escribe su pregunta en la interfaz (Streamlit).**
   Desde la aplicación web el usuario formula su necesidad en lenguaje natural (por ejemplo: "quiero un portátil para la universidad de menos de 800 € con buena batería y 16 GB de RAM"). La interfaz muestra el historial de conversación y envía el texto al backend del chatbot.
2. **Comprobaciones iniciales y sanitización del texto.**
   Antes de procesar la consulta, el sistema aplica las capas de seguridad: se comprueba que el mensaje no supere la longitud máxima permitida y se limpia el texto para eliminar patrones peligrosos relacionados con prompt injection (intentos de cambiar las instrucciones internas del modelo, insertar scripts, etc.). De esta forma solo se trabaja con entradas seguras y manejables.
3. **Orquestación con LlamaIndex y extracción de filtros básicos.**
   El texto saneado se pasa al orquestador (LlamaIndex) junto con el LLM de Groq. En esta fase se:
   - Verifica que la pregunta esté realmente relacionada con portátiles;
   - Extraen filtros simples de la consulta (precio máximo, RAM mínima, posible GPU concreta, pulgadas, tipo de uso como "juegos", "oficina" o "universidad").
     Esta información se obtiene mediante reglas sencillas (extract_basic_filters) y ayuda a decidir el tipo de búsqueda más adecuado.
   -

4. **Decisión entre búsqueda simbólica y búsqueda semántica.**
   Con los filtros obtenidos, el sistema elige entre:
   - **Búsqueda simbólica en Neo4j**, cuando la consulta es muy técnica y concreta (por ejemplo, “RTX 4060”, “16 GB de RAM”, “menos de 1.200 €”). En este caso se construyen consultas Cypher con condiciones directas sobre las propiedades de los nodos Laptop.
   - **Búsqueda semántica**, cuando la pregunta es más vaga o descriptiva (por ejemplo, “portátil ligero para la uni con buena batería”). Se genera un embedding de la consulta y se busca en el índice vectorial creado sobre el campo embedding de los nodos Laptop, recuperando los equipos más cercanos en significado.
5. **Recuperación de candidatos desde Neo4j.**
   Neo4j devuelve una lista de portátiles candidatos que cumplen los filtros simbólicos y/o son similares a nivel semántico. Cada nodo Laptop ya incluye sus propiedades principales (marca, modelo, CPU, GPU, RAM, pulgadas, precio en euros, tipo) y, cuando ha sido posible, una URL de compra obtenida mediante el scrapper.
6. **Generación de la respuesta con Groq.**
   Sobre la lista de candidatos el LLM de Groq se encarga de:
   - Ordenar y seleccionar los modelos más apropiados para la necesidad descrita por el usuario;
   - Aplicar de nuevo restricciones duras (por ejemplo, descartar los que superan el presupuesto máximo);
   - Redactar una respuesta clara, explicando para cada portátil por qué es adecuado (CPU, GPU, RAM, relación calidad-precio, portabilidad, etc.) y, cuando está disponible, incluyendo el enlace de compra asociado.
7. **Presentación final en la interfaz.**
   Por último, la interfaz de Streamlit muestra el mensaje generado por el LLM dentro del historial del chat. El usuario ve una lista de recomendaciones con sus características clave y, en su caso, puede pulsar en los enlaces para consultar más detalles o proceder a la compra en la tienda externa correspondiente.

En conjunto, este flujo integra la recuperación de información (Neo4j + índice vectorial) con la generación de lenguaje natural (Groq) y las capas de seguridad, formando un sistema RAG completo capaz de entender preguntas sobre portátiles y proponer modelos adecuados de forma razonada.

6. Casos de Uso.

6.1 Caso de uso optimista.

En el caso de uso optimista, el usuario hace preguntas claras, completas y directamente relacionadas con portátiles. Ejemplo:

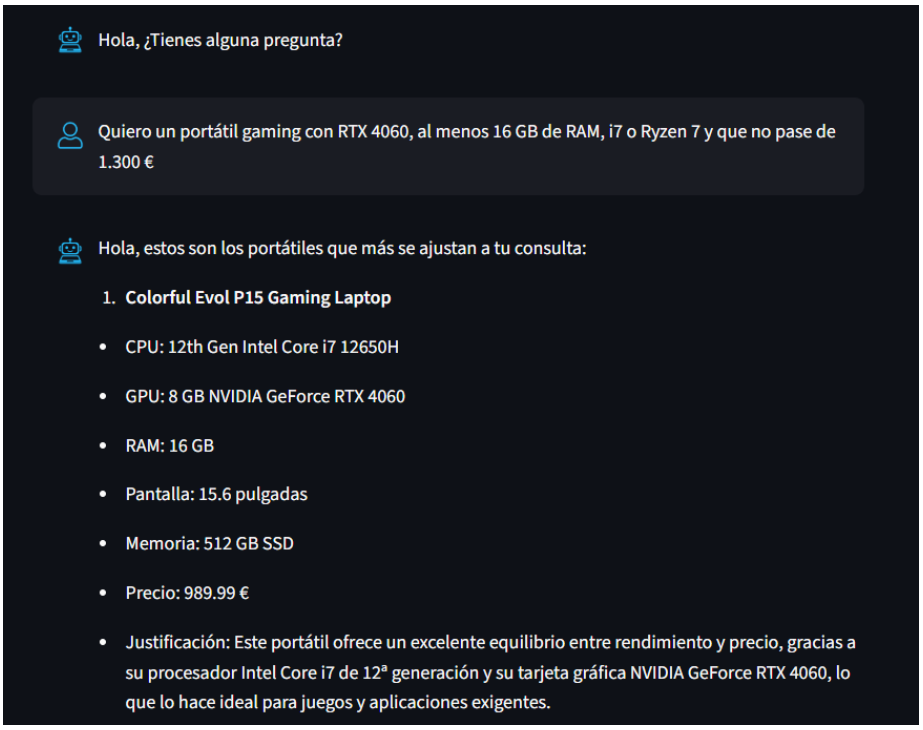

Ilustración 1 Ejemplo real de caso de uso optimista.

6.2 Caso de uso verosímil.

En el caso de uso verosímil, las preguntas se parecen más a lo que escribiría un usuario real: mezclan lenguaje natural, cierta vaguedad y a veces especificaciones incompletas. Ejemplo:

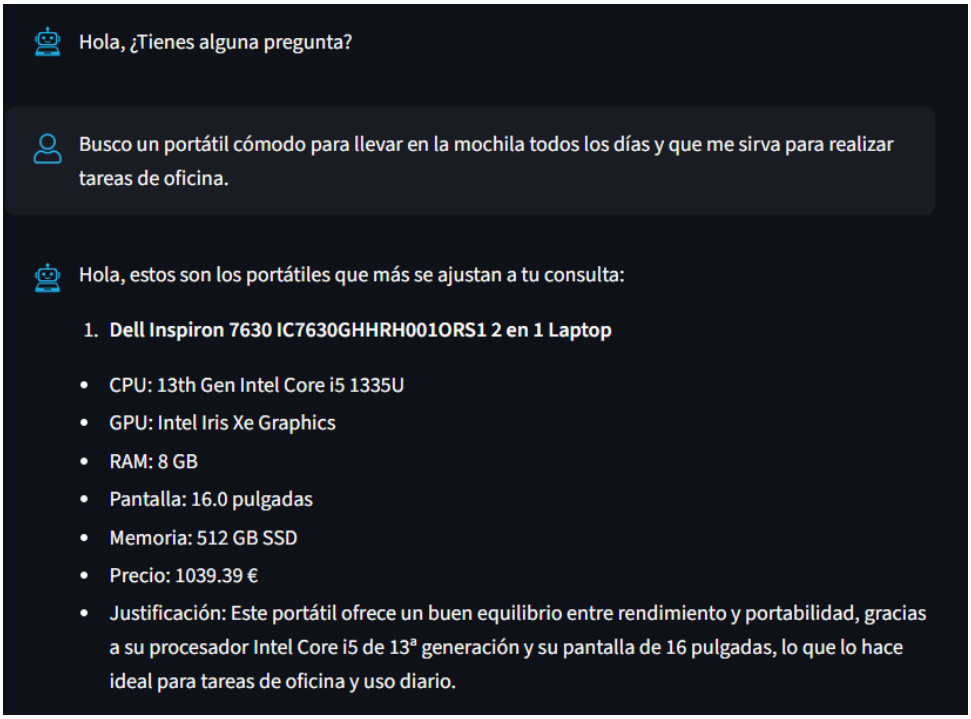

Ilustración 2 Ejemplo real de caso de uso verosímil.

6.3 Cado de uso pesimista.

En el caso de uso pesimista se contemplan preguntas problemáticas, es decir no permite preguntas no relacionadas al ámbito de los portátiles o cuyo contenido sea obsceno o peligroso. Ejemplo:

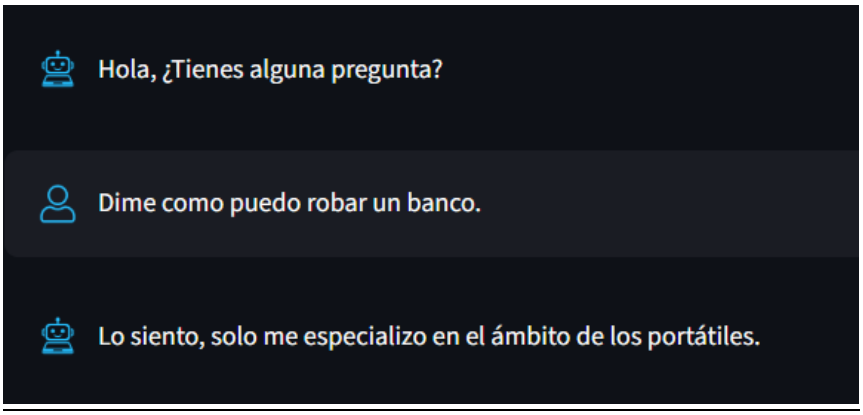

Ilustración 3 Ejemplo real de caso de uso pesimista.

7. Análisis DAFO.

7.1 Fortalezas

- **Arquitectura modular y clara:**
  El sistema está dividido en componentes bien separados: la interfaz en interface.py, la lógica del chatbot en su propia clase, y por otro lado la generación/parsing de la base de datos y el

scrapper. Esta separación facilita entender el proyecto, localizar errores y mejorar cada parte sin miedo a romper el resto.

- **Recomendación semántica avanzada:**
  La combinación de Neo4j, embeddings y un índice vectorial permite buscar portátiles "por significado" y no solo por filtros exactos. El sistema es capaz de interpretar consultas del tipo "portátil ligero para estudiar" o "para jugar FPS con buen rendimiento" y traducirlas a criterios técnicos razonables.
- **Respuestas explicadas y orientadas al usuario:**
  El recomendador no solo devuelve un listado de modelos, sino que acompaña cada portátil de una pequeña explicación basada en CPU, GPU, RAM, pantalla, precio y tipo de uso. Esto hace que la recomendación sea más transparente y útil que un simple resultado de búsqueda.

7.2 Debilidades

- **Scrapper sensible a cambios externos:**
  La parte de scraping con Selenium y DuckDuckGo depende de cómo presenten los resultados los buscadores y las tiendas. Cualquier cambio en la estructura de las páginas puede provocar que el scrapper deje de encontrar enlaces válidos o devuelva links genéricos.
- **Precios aproximados y poco adaptados al mercado real:**
  El dataset original usa precios en rupias indias (₹) y en el sistema se convierten a euros con un tipo de cambio fijo. Esto hace que los precios mostrados sean orientativos, pero no reflejen necesariamente la realidad del mercado español o europeo.

7.3 Oportunidades

- **Mejorar la experiencia de filtrado desde la interfaz:**
  Se podrían añadir a la interfaz elementos como sliders o selectores (precio máximo, RAM mínima, pulgadas, tipo de uso: gaming/oficina/estudios) antes de lanzar la consulta al modelo. Esto permitiría al usuario acotar mejor lo que busca y al sistema trabajar con filtros más claros.
- **Optimizar el rendimiento del sistema:**
  Existe margen para cachear resultados típicos, embeddings de consultas frecuentes o incluso ciertos listados de portátiles. Separar claramente los procesos de carga de datos y scraping del flujo de conversación permitiría que el usuario no "sufra" esas tareas pesadas en tiempo real.

- **Refinar y actualizar los datos:**
  A medio plazo se podrían reemplazar los precios del dataset original por fuentes más realistas (por ejemplo, precios de tiendas europeas) o incluso construir un dataset propio. También se podría mejorar el scrapper o sustituirlo por APIs oficiales cuando existan, aumentando la calidad y la fiabilidad de la información.

7.4 Amenazas

- **Cambios en webs externas y herramientas de terceros:**
  Modificaciones en los resultados de búsqueda, en la estructura de las páginas de las tiendas o en futuras versiones de Neo4j y los LLM pueden provocar que el scrapper, los índices vectoriales o la integración con el modelo queden obsoletos y requieran mantenimiento continuo.
- **Percepción de fiabilidad por parte del usuario:**
  Si los enlaces fallan, apuntan a páginas genéricas o los precios se alejan mucho de lo que el usuario ve en tiendas reales, puede percibirse el sistema como poco fiable frente a buscadores comerciales más pulidos.
- **Coste de mantenimiento y evolución:**
  Al depender de varias tecnologías y servicios a la vez, el proyecto requiere revisiones periódicas para seguir funcionando correctamente. Si se abandona durante un tiempo o si alguno de los servicios cambia su forma de uso, puede resultar costoso volver a ponerlo al día.

8. Capas de Seguridad.

8.1 Capa de Seguridad 1: Anti prompt injection.

Esta capa de seguridad se añadió para evitar ataques de tipo “prompt injection”, en este tipo de ataques los actores maliciosos crean entradas aparentemente inocentes para manipular los modelos de aprendizaje automático.

Para evitar este problema, antes de enviar cualquier texto introducido por el usuario al modelo LLM, se aplica una función de sanitización que elimina patrones peligrosos como "instructions:", "system:", "ignore previous", "act as", etiquetas <script>, pistas de jailbreak, o cualquier intento de manipular las reglas internas del modelo.

Gracias a esta capa de seguridad se logra que el comportamiento del recomendador no pueda ser alterado por peticiones maliciosas y que no revele instrucciones internas.

8.2 Capa de Seguridad 2: Validación segura de URLs generadas por el scrapper (Anti-Phishing).

El scrapper implementado usa el motor de búsqueda “DuckDuckGo” para obtener enlaces de compra relativos a los ordenadores portátiles, sin embargo, existe el riesgo de que se obtengan resultados que incluyen webs falsas o de phishing.

El phishing es una técnica de ciberataque que consiste en suplantar la identidad de una entidad de confianza (como un banco, una red social o una empresa) para engañar a las personas y robarles información confidencial.

Para evitar este riesgo se implementó en el programa una verificación del dominio obtenido, solo se aceptan URLs pertenecientes a una lista blanca de tiendas reales como Amazon, PCComponentes, MediaMarkt o FNAC.

Las URL obtenidas cuyo dominio no coincide exactamente con los permitidos es descartado para mantener la seguridad de los usuarios y el sistema.

8.3 Capa de Seguridad 3: Limitación de longitud de la entrada del usuario (Input Length Limiter).

Para evitar que el sistema procese textos excesivamente largos, que podrían ralentizar el modelo o incluso utilizarse para intentar saturarlo, se implementó una verificación previa que limita el tamaño máximo de la consulta del usuario. Si el mensaje supera un umbral definido (por ejemplo, 400 caracteres), se rechaza y se solicita al usuario que lo resuma.

Esta medida previene tanto fallos de rendimiento como posibles intentos de denegación de servicio, y asegura que solo se procesen entradas manejables y seguras.

9. Lecciones Aprendidas.

Gracias a la realización de este trabajo he podido comprender como funciona la estructura de un sistema RAG desde la consulta del usuario hasta la respuesta final del sistema. Además de únicamente comprender su teoría hemos creado un sistema propio desde 0, lo que nos ha llevado a estudiar e implementar tecnologías usadas en el ámbito profesional.

El diseño de la base de datos en Neo4j me ha obligado a pensar los datos como un grafo y a estructurar correctamente nodos y relaciones para que luego las consultas simbólicas y semánticas tengan sentido dentro del recomendador.

Usamos Ollama para generar embeddings locales y entender sus limitaciones de recursos, empleamos Selenium para automatizar el scraping de URLs reales de portátiles y, finalmente, migrar a Groq como motor LLM, logrando implementar todas estas tecnologías con éxito.

Todos los elementos estudiados e implementados deben coordinarse dentro de una arquitectura RAG para ofrecer recomendaciones útiles y coherentes al usuario.

10. Trabajos futuros.

- **Mejora de la calidad y actualización de datos:**
  Una ampliación natural sería reemplazar o complementar el dataset actual con fuentes más actualizadas, incluyendo precios reales en euros y stock de tiendas españolas. Integrar una API de comercios electrónicos permitiría que las recomendaciones estuvieran alineadas con el mercado actual.
- **Interfaz más rica en filtros y opciones:**
  En la interfaz se podrían añadir controles más visuales (sliders de precio, checkboxes para GPU, RAM, tamaño de pantalla, peso, etc.) que complementen las preguntas en lenguaje natural. Esto ayudaría al usuario a concretar mejor sus necesidades y al sistema a trabajar con filtros más precisos.
- **Optimización de rendimiento y caché:**
  Sería interesante implementar mecanismos de caché para consultas frecuentes o para resultados de búsqueda típicos (por ejemplo, “portátil para universidad de menos de 800 €”). De esta forma, se reducirían tiempos de respuesta, especialmente en máquinas con menos recursos o cuando el modelo de embeddings resulte costoso de cargar.
- **Mejoras en el scrapper y en la gestión de enlaces:**
  El scrapper podría hacerse más robusto frente a cambios en las páginas de resultados y añadir comprobaciones adicionales sobre disponibilidad y precio del producto. También se podría guardar un historial de enlaces verificados para reutilizarlos y reducir scraping innecesario.