TRABAJO PRÁCTICO FINAL

PROCESAMIENTO DE LENGUAJE NATURAL

Rehecho

Integrantes: Cena Lautaro

Fecha limite: 26/12

Año: 2025

1.1 Introducción y objetivo del trabajo

El presente trabajo práctico tiene como objetivo el diseño e implementación de un sistema de recuperación y consulta de información para un contexto de **atención al cliente de una empresa dedicada a la comercialización de electrodomésticos**, que opera con múltiples líneas de productos, sucursales y canales de venta, utilizando técnicas de Procesamiento de Lenguaje Natural (NLP) y modelos de lenguaje de gran escala (LLM).

En este tipo de organizaciones, la dinámica de atención al cliente implica responder, de manera rápida y precisa, a consultas muy diversas: desde preguntas sobre el uso y mantenimiento de un producto específico, hasta solicitudes de información estructurada sobre stock disponible, precios, ventas, devoluciones o relaciones entre productos, categorías, marcas y sucursales. Estas consultas pueden provenir tanto de clientes finales como de personal interno (ventas, soporte o logística), y suelen formularse en lenguaje natural, sin una estructura predefinida.

El problema central que se busca resolver es, entonces, la **heterogeneidad tanto de la información disponible como de las consultas de los usuarios**. En un escenario realista, un asistente inteligente debe ser capaz de interpretar correctamente la intención detrás de cada consulta y acceder a la fuente de datos más adecuada según el tipo de información requerida.

Una solución basada en una única fuente de datos o en un único tipo de base resulta insuficiente para cubrir este espectro de necesidades. Por este motivo, el sistema propuesto adopta una **arquitectura híbrida**, que combina distintos tipos de bases de datos y estrategias de recuperación, coordinadas mediante un clasificador de intención y reforzadas, cuando corresponde, por un proceso de reranking basado en modelos de lenguaje.

1.2 Motivación del uso de múltiples bases de datos

Uno de los principales desafíos abordados en este trabajo es que **no todas las preguntas se responden del mismo modo**, ni requieren el mismo tipo de información subyacente.

Al analizar el dominio del problema, se identificaron tres grandes tipos de consultas:

1. Consultas semánticas y descriptivas, típicas de usuarios finales, que requieren interpretar lenguaje natural y recuperar fragmentos de texto relevantes.

2. Consultas estructuradas, que solicitan listados, filtros o agregaciones sobre datos tabulares.

3. Consultas relacionales, que exploran vínculos entre entidades del dominio (por ejemplo, productos y categorías, productos y marcas, productos y motivos de devolución).

Cada uno de estos tipos de consultas presenta requerimientos distintos en términos de modelado de datos y mecanismos de recuperación. Forzar todas las consultas a un único paradigma implicaría pérdida de precisión, mayor complejidad de implementación o necesidad de parseos manuales poco robustos.

Por ello, el sistema se apoya en **tres bases complementarias**, cada una optimizada para un tipo específico de información y consulta.

1.3 Base de datos vectorial

La base de datos vectorial se utiliza para resolver consultas de tipo semántico, es decir, aquellas en las que el usuario formula preguntas en lenguaje natural relacionadas con el uso, funcionamiento, características o problemas asociados a los productos comercializados por la empresa.

En este trabajo, la base vectorial se construye a partir de dos fuentes principales de información no estructurada:

- **Manuales de productos**, que contienen descripciones técnicas, instrucciones de uso, mantenimiento y especificaciones.

- **Reseñas de usuarios**, que aportan información empírica sobre el uso real de los productos, problemas frecuentes y percepciones generales.

Ambos tipos de documentos son previamente procesados mediante una etapa de **fragmentación en chunks**, con el objetivo de dividir los textos largos en segmentos manejables y semánticamente coherentes. Cada fragmento es luego transformado en un **embedding vectorial**, lo que permite representar su contenido semántico en un espacio numérico de alta dimensión.

La recuperación de información se realiza mediante **búsqueda por similitud semántica**, comparando el embedding de la consulta del usuario con los embeddings de los chunks almacenados. Este enfoque resulta especialmente adecuado para consultas abiertas o mal estructuradas, donde no existe una correspondencia directa con un campo tabular ni una relación explícita de tipo grafo.

Ejemplos típicos de consultas que se resuelven a través de la base vectorial incluyen:

- Problemas de uso o funcionamiento de un producto.

- Procedimientos de mantenimiento o limpieza.

- Interpretación de mensajes de error.

- Descripciones y características generales de los productos.

La base vectorial constituye el núcleo del enfoque **RAG (Retrieval-Augmented Generation)** adoptado en el sistema. Los fragmentos recuperados mediante similitud semántica no se utilizan directamente, sino que son posteriormente refinados mediante un proceso de **reranking basado en LLM**, con el objetivo de priorizar aquellos chunks más relevantes para la consulta específica del usuario.

1.4 Base de datos tabular

La base de datos tabular se emplea para responder consultas que requieren acceder a información estructurada, como stock, precios, ventas, clientes, devoluciones o sucursales.

En el desarrollo del trabajo práctico, estos datos se encuentran organizados en **DataFrames de Pandas**, que representan distintas tablas del dominio del problema (por ejemplo, productos,

inventario, ventas y devoluciones). Este formato permite realizar operaciones eficientes de filtrado, agregación y análisis sobre grandes volúmenes de información estructurada.

La base tabular resulta especialmente adecuada para:

- Consultas de conteo y agregación (por ejemplo, cantidades vendidas o devoluciones totales).
- Filtros por columnas específicas (producto, sucursal, cliente, fecha, etc.).
- Consultas temporales o geográficas.
- Generación de reportes estructurados.

Un aspecto central del diseño implementado en este trabajo es que las consultas tabulares **no se resuelven mediante parseo manual de strings ni mediante la ejecución de código arbitrario generado dinámicamente**. En su lugar, el sistema utiliza un modelo de lenguaje para generar **filtros estructurados en formato JSON**, que describen explícitamente las condiciones a aplicar sobre las columnas del DataFrame.

Estos filtros JSON son luego traducidos a **máscaras booleanas seguras**, que se aplican directamente sobre los DataFrames de Pandas. Este enfoque aporta varias ventajas relevantes:

- Mejora la robustez y mantenibilidad del sistema.
- Reduce el riesgo de errores o comportamientos inesperados.
- Evita problemas de seguridad asociados a la ejecución de código dinámico.
- Facilita la integración con otros componentes del sistema, en particular con la base de grafos, permitiendo restringir consultas tabulares a subconjuntos de entidades previamente obtenidas desde el grafo.

De esta manera, la base tabular cumple un rol fundamental en la resolución de consultas estructuradas dentro de la arquitectura híbrida propuesta.

1.5 Base de datos de grafos

La base de datos de grafos se utiliza para modelar y consultar relaciones entre entidades del dominio que, si bien podrían representarse de forma relacional, resultan mucho más naturales y expresivas cuando se describen como un grafo.

En el contexto de una tienda de electrodomésticos, el sistema debe responder preguntas que no se limitan a atributos aislados de una entidad, sino que involucran **relaciones múltiples y recorridos entre nodos**. Ejemplos típicos de este tipo de consultas incluyen la exploración de jerarquías de productos, asociaciones entre marcas y categorías, o el análisis de patrones de devoluciones.

En este trabajo, dichas relaciones se modelan explícitamente utilizando **Neo4j**, donde cada entidad relevante se representa como un nodo, y las relaciones entre ellas como aristas tipadas. Entre las relaciones implementadas se encuentran, entre otras:

- Producto → Categoría
- Producto → Subcategoría
- Producto → Marca
- Producto → Motivo de devolución
- Producto → Sucursal

Este esquema permite capturar de forma directa la semántica del dominio, evitando la necesidad de joins complejos, tablas intermedias o lógica relacional dispersa en múltiples consultas SQL.

Gracias a esta representación, las consultas de tipo grafo permiten responder de manera eficiente y legible preguntas como:

- Qué productos pertenecen a una categoría o subcategoría determinada.
- Qué marcas están asociadas a productos con ciertas características.
- Qué categorías concentran mayor cantidad de devoluciones.
- Qué productos están vinculados a determinados motivos de reclamo o garantía.

En particular, el uso de Neo4j y del lenguaje de consultas **Cypher** facilita la exploración de conexiones y patrones relacionales, permitiendo expresar consultas complejas de manera declarativa y alineada con la estructura conceptual del problema.

Además, la base de grafos no funciona de manera aislada, sino que se integra con los demás componentes del sistema. En ciertos flujos de consulta, los resultados obtenidos desde el grafo (por ejemplo, listas de identificadores de productos relacionados) se utilizan posteriormente para restringir búsquedas en la base tabular, logrando así una combinación efectiva entre análisis relacional y datos estructurados.

En síntesis, la incorporación de una base de datos de grafos aporta expresividad, claridad conceptual y flexibilidad al sistema, siendo un componente clave para resolver consultas relacionales complejas dentro del asistente inteligente propuesto.

Sí, **está bien y es esperable** que en tu comparación el **clasificador tradicional rinda igual o incluso mejor que el basado en LLM** en este TP. No hay nada “anómalo” ahí.
Te explico primero **por qué**, y después te dejo **el texto corregido y ampliado** del punto 1.6, incorporando métricas y comparación **sin números** como pediste.

1.6 Clasificador de intención como coordinador del sistema

Dado que el sistema integra múltiples fuentes de información y estrategias de recuperación, resulta indispensable contar con un mecanismo que determine cómo debe procesarse cada consulta del usuario. No todas las preguntas pueden resolverse de la misma manera ni sobre la misma base de datos, y aplicar un enfoque incorrecto impacta directamente en la calidad, precisión y utilidad de la respuesta final.

Para abordar este problema, se implementa un **clasificador de intención**, cuya función es asignar cada consulta a una de las siguientes categorías:

- **Vectorial**: consultas semánticas en lenguaje natural relacionadas con uso, funcionamiento, problemas, procedimientos o características descriptivas de los productos.
- **Tabular**: consultas estructuradas sobre información cuantitativa o categórica, como stock, ventas, precios, clientes o sucursales.
- **Grafo**: consultas que involucran relaciones entre entidades del dominio, tales como producto–categoría, producto–marca, producto–sucursal o motivos de devolución.

Este clasificador actúa como el **orquestador central del sistema**, desacoplando la lógica de decisión del resto de los módulos. A partir de su predicción, la consulta se enruta automáticamente hacia el pipeline correspondiente, evitando el uso de reglas rígidas, heurísticas manuales o flujos de control difíciles de mantener y escalar.

En el trabajo se implementan y evalúan **dos enfoques complementarios** para la clasificación de intención:

- **Clasificador tradicional**, basado en:
    - Vectorización TF-IDF con unigramas y bigramas.
    - Modelo de regresión logística.
    - Entrenamiento supervisado sobre un conjunto de ejemplos representativos de cada intención.
- **Clasificador basado en LLM**, que utiliza:
    - Un prompt explícito con definición clara de las clases.
    - Ejemplos ilustrativos de cada tipo de consulta.
    - Inferencia directa de la intención a partir del texto de entrada, sin entrenamiento adicional.

Ambos enfoques se evalúan sobre un **mismo conjunto de test**, utilizando métricas estándar de clasificación multiclase, como *accuracy*, *precision*, *recall* y *F1-macro*, además de matrices de confusión para analizar los patrones de error.

Los resultados muestran que el **clasificador tradicional presenta un desempeño muy competitivo e incluso superior en algunas métricas**, lo cual resulta esperable dado el carácter acotado del dominio y la claridad semántica de las clases. Este comportamiento evidencia que, para tareas bien definidas y con datasets pequeños, los modelos clásicos continúan siendo una solución sólida y eficiente.

Por su parte, el clasificador basado en LLM muestra un comportamiento más flexible, aunque con una mayor tendencia a confundir intenciones cercanas (especialmente entre consultas vectoriales y tabulares). No obstante, su principal fortaleza reside en su **capacidad de generalización conceptual**, su menor dependencia del entrenamiento supervisado y su facilidad de adaptación a nuevos dominios o clases sin necesidad de reentrenar el modelo.

En conjunto, esta comparación no busca establecer un “ganador” absoluto, sino **analizar empíricamente las ventajas y limitaciones de cada enfoque** dentro del contexto del sistema propuesto. La coexistencia de ambos modelos refuerza la arquitectura modular del sistema y habilita decisiones informadas sobre qué estrategia resulta más adecuada según el escenario de uso.

1.7 Reranking de resultados vectoriales mediante modelos de lenguaje

En el pipeline de recuperación vectorial, la búsqueda inicial se realiza a partir de embeddings, recuperando los fragmentos de texto (chunks) más similares a la consulta del usuario según una métrica de similitud semántica. Si bien este enfoque resulta eficiente y escalable, presenta

una limitación conocida: la similitud vectorial no siempre refleja con precisión la **relevancia contextual** de un fragmento respecto de una consulta específica.

En el trabajo, esta situación se observa especialmente en consultas donde varios fragmentos comparten vocabulario o temática general, pero solo algunos contienen información verdaderamente útil para responder la pregunta planteada. En estos casos, el ranking inicial basado únicamente en embeddings puede devolver resultados parcialmente relevantes o mal ordenados.

Para mitigar este problema, se incorpora un **reranker basado en modelos de lenguaje (LLM)**, que actúa como una segunda etapa de evaluación sobre los candidatos recuperados por la búsqueda vectorial.

Rol del reranker en la arquitectura

El reranker no reemplaza a la base vectorial, sino que la **complementa**. Su función es:

- Tomar un conjunto reducido de fragmentos candidatos (top-k recuperados por similitud).
- Reanalizar cada fragmento considerando explícitamente la consulta original.
- Asignar un **score de relevancia adicional**, basado en razonamiento semántico y contextual.
- Reordenar los fragmentos priorizando aquellos más adecuados para generar una respuesta final.

De este modo, el sistema combina:

- **Eficiencia** en la recuperación inicial (embeddings).
- **Precisión contextual** en la etapa final (LLM).

Implementación adoptada en el trabajo

En la implementación original del TP, el reranking se realizaba mediante **múltiples llamadas al LLM**, evaluando cada fragmento de manera individual. Si bien este enfoque resulta conceptualmente sencillo, presenta dos desventajas principales:

- **Costo computacional elevado**, al requerir una llamada al modelo por cada chunk.
- **Mayor latencia**, especialmente cuando el número de fragmentos candidatos aumenta.

A partir de la corrección, este diseño se optimiza para que el reranker realice la evaluación de **todos los fragmentos candidatos en una única llamada al LLM**. En este esquema:

- La consulta del usuario y la lista de fragmentos se incluyen conjuntamente en el prompt.
- El modelo devuelve un ranking completo o un conjunto de scores en una sola respuesta.
- El sistema reordena los fragmentos utilizando esta información, sin iteraciones adicionales.

Esta optimización reduce significativamente el número de llamadas al modelo, manteniendo la calidad del reranking y mejorando la eficiencia general del pipeline.

Beneficios del reranking con LLM

La incorporación del reranker aporta mejoras concretas al sistema:

- Permite distinguir entre fragmentos semánticamente similares pero contextualmente distintos.
- Mejora la calidad de las respuestas generadas en consultas complejas o ambiguas.
- Reduce la probabilidad de que fragmentos irrelevantes aparezcan en las primeras posiciones.
- Refuerza el enfoque RAG, al asegurar que la generación se base en contexto realmente pertinente.

En particular, el reranking resulta clave cuando los documentos contienen información repetida, descripciones generales o menciones tangenciales al problema consultado, situaciones frecuentes en manuales técnicos y reseñas de usuarios.

Consideraciones de diseño

El reranker se aplica **exclusivamente a consultas de tipo vectorial**, ya que en los casos tabulares y de grafo la noción de ranking semántico no resulta pertinente. Esta decisión mantiene la coherencia del sistema y evita introducir complejidad innecesaria en pipelines donde el orden de los resultados no depende del lenguaje natural.

En conjunto, el reranking por LLM constituye una mejora cualitativa del sistema, alineada con escenarios reales de atención al cliente, donde la relevancia contextual de la información es tan importante como su disponibilidad.

1.8 Pipeline de recuperación de información

El pipeline de recuperación constituye el núcleo operativo del sistema, y define el recorrido completo que sigue una consulta del usuario desde su ingreso hasta la obtención de resultados relevantes, antes de cualquier etapa de generación o presentación final.

El proceso comienza cuando el usuario formula una consulta en lenguaje natural. Esta consulta es analizada por el **clasificador de intención**, que determina si debe ser tratada como una

consulta de tipo vectorial, tabular o grafo. A partir de esta decisión, la consulta se enruta automáticamente hacia el pipeline de recuperación correspondiente.

En el caso de las **consultas vectoriales**, la recuperación se realiza en dos etapas. Primero, la consulta se transforma en un embedding y se ejecuta una búsqueda por similitud sobre la base vectorial construida a partir de manuales de productos y reseñas de usuarios. Esta etapa devuelve un conjunto inicial de fragmentos (chunks) candidatos, ordenados según su similitud semántica. Dado que esta métrica no siempre refleja la relevancia contextual completa, se aplica posteriormente un **reranking basado en LLM**, que reevalúa los fragmentos considerando conjuntamente la consulta y el contenido textual, asignando un score adicional y produciendo un ranking final más preciso.

Para las **consultas tabulares**, el pipeline adopta un enfoque distinto. La consulta del usuario se procesa mediante un LLM que genera una especificación de filtros estructurados en formato JSON. Estos filtros describen condiciones sobre columnas concretas (igualdades, rangos, pertenencia a conjuntos, coincidencias textuales, etc.) y se traducen directamente en máscaras booleanas sobre los DataFrames de Pandas. De este modo, la recuperación de resultados se realiza de forma segura, reproducible y sin ejecutar código generado dinámicamente, evitando parseos manuales de texto o mecanismos frágiles.

En las **consultas de tipo grafo**, el pipeline opera sobre la base de datos de grafos (Neo4j), donde se modelan explícitamente las relaciones entre entidades del dominio. A partir de la consulta, se recuperan conjuntos de nodos o identificadores (por ejemplo, productos, categorías o marcas) que cumplen con la relación solicitada. Estos resultados pueden devolverse directamente o bien utilizarse como restricción adicional para otros pipelines, por ejemplo limitando una consulta tabular a un subconjunto de productos obtenido desde el grafo.

En todos los casos, el pipeline de recuperación se encuentra claramente desacoplado del resto del sistema. Cada tipo de consulta sigue un flujo bien definido, especializado y coherente con la naturaleza de los datos involucrados. Esta separación permite optimizar cada estrategia de recuperación de manera independiente, mejorar la interpretabilidad del sistema y facilitar su extensión futura.

Hasta este punto, el sistema se enfoca exclusivamente en la **identificación, recuperación y ordenamiento de la información relevante**, dejando las etapas posteriores de integración, presentación y evaluación para los capítulos siguientes.

1.9 Integración para generación y conversación

Una vez finalizada la etapa de recuperación de información, el sistema integra los resultados obtenidos para construir una respuesta final orientada a la interacción con el usuario. Esta fase

no se limita a devolver datos crudos, sino que busca articular la información recuperada en un formato comprensible y coherente, utilizando modelos de lenguaje como capa de generación.

La forma en que se integra la información depende del tipo de pipeline activado previamente:

En las **consultas vectoriales**, los fragmentos recuperados y rerankeados se utilizan como contexto para el modelo de lenguaje. El LLM recibe la consulta original junto con los chunks más relevantes y genera una respuesta en lenguaje natural apoyada en esa evidencia textual. De este modo, el sistema adopta un enfoque de tipo *Retrieval-Augmented Generation (RAG)*, donde la generación está explícitamente condicionada por documentos reales del dominio (manuales y reseñas), reduciendo alucinaciones y mejorando la precisión de las respuestas.

En las **consultas tabulares**, el sistema no genera respuestas a partir de texto libre, sino que primero ejecuta los filtros estructurados sobre los DataFrames y obtiene un subconjunto de resultados concretos. Estos resultados pueden luego presentarse directamente o bien resumirse mediante el LLM, que transforma la salida estructurada en una respuesta conversacional, manteniendo la fidelidad a los datos subyacentes.

Para las **consultas de tipo grafo**, los resultados consisten en entidades y relaciones explícitas (por ejemplo, conjuntos de productos, categorías o marcas). Estos resultados se integran de manera similar, ya sea devolviéndolos como listados claros o utilizándolos como contexto adicional para una respuesta explicativa generada por el modelo de lenguaje.

Un aspecto importante del diseño es que la **generación de la respuesta se mantiene desacoplada de la lógica de recuperación**. El sistema primero decide *qué información es relevante* y *desde dónde obtenerla*, y recién luego utiliza el LLM para formular una respuesta adecuada al usuario. Este enfoque evita que el modelo de lenguaje deba inferir o “adivinar” información estructural, y lo posiciona como un componente de síntesis y comunicación, más que como una fuente primaria de conocimiento.

En conjunto, esta capa de integración permite transformar los distintos tipos de resultados —textuales, tabulares y relacionales— en una experiencia conversacional unificada, coherente con el contexto del usuario y alineada con los objetivos de un sistema real de atención al cliente.

---

2. Ejecución del sistema y resolución de consultas

Este capítulo describe el funcionamiento del sistema completo a través de la ejecución de consultas reales, mostrando cómo los distintos módulos desarrollados interactúan entre sí para producir una respuesta final. A diferencia del capítulo anterior, centrado en la arquitectura y el diseño, aquí el énfasis está puesto en el **flujo operativo del sistema**.

2.1 Flujo general de procesamiento de una consulta

Ante una consulta formulada por el usuario en lenguaje natural, el sistema ejecuta las siguientes etapas:

1. Recepción de la consulta.
2. Clasificación de intención.
3. Enrutamiento hacia el pipeline correspondiente.
4. Recuperación de información desde la base adecuada.
5. (Cuando corresponde) Reranking de resultados mediante LLM.
6. Integración de los resultados y generación de la respuesta.

Este flujo se mantiene constante para todas las consultas, independientemente de su tipo, lo que permite un comportamiento consistente y fácilmente extensible.

2.2 Clasificación de intención y enrutamiento

El primer paso operativo consiste en clasificar la intención de la consulta. Tal como se describió en el capítulo anterior, el sistema distingue entre tres tipos de intención: vectorial, tabular y grafo.

En esta etapa, la consulta se analiza sin acceder aún a ninguna base de datos. El clasificador actúa como un **filtro inicial**, evitando ejecutar pipelines innecesarios y reduciendo el costo computacional del sistema.

Dependiendo de la intención predicha, la consulta se enruta automáticamente hacia uno de los siguientes módulos:

- Pipeline vectorial (RAG).
- Pipeline tabular basado en DataFrames.
- Pipeline de consultas relacionales sobre el grafo.

Este desacople entre decisión y ejecución permite que cada pipeline se mantenga independiente y especializado.

2.3 Ejecución de consultas vectoriales (pipeline RAG)

Cuando la intención detectada es **vectorial**, la consulta se procesa mediante un pipeline de recuperación semántica.

En primer lugar, la consulta se transforma en un embedding y se utiliza para buscar fragmentos relevantes dentro de la base vectorial, construida a partir de manuales de productos y reseñas de usuarios previamente fragmentados en chunks.

La búsqueda inicial devuelve un conjunto de candidatos basados en similitud semántica. Sin embargo, dado que esta métrica puede priorizar fragmentos parcialmente relevantes, el sistema incorpora una etapa adicional de **reranking mediante LLM**.

En esta etapa, el modelo de lenguaje evalúa cada fragmento candidato en función de la consulta original y asigna un score de relevancia adicional. Luego, los fragmentos se ordenan considerando este score, lo que permite seleccionar los contextos más adecuados para la generación de la respuesta.

Finalmente, los fragmentos mejor rankeados se utilizan como contexto para el modelo de lenguaje, que genera una respuesta en lenguaje natural basada exclusivamente en la información recuperada, siguiendo un enfoque RAG.

2.4 Ejecución de consultas tabulares

Cuando la consulta es clasificada como **tabular**, el sistema accede a los DataFrames que representan la información estructurada del dominio (productos, inventario, ventas, devoluciones, etc.).

En lugar de interpretar la consulta mediante reglas manuales o ejecutar código arbitrario, el sistema utiliza un modelo de lenguaje para **traducir la consulta del usuario a un conjunto de filtros estructurados en formato JSON**. Estos filtros describen operaciones simples (comparaciones, pertenencia, búsquedas de texto) sobre columnas específicas.

Dichos filtros se aplican de manera segura mediante máscaras booleanas sobre los DataFrames de Pandas, garantizando que la ejecución sea controlada y reproducible.

Este enfoque permite responder consultas complejas (por ejemplo, combinando condiciones de stock, ubicación y producto) sin comprometer la seguridad ni la claridad del sistema, y facilita la integración con otros módulos, como el grafo.

2.5 Ejecución de consultas de tipo grafo

Las consultas clasificadas como **grafo** se orientan a explorar relaciones entre entidades del dominio. Estas relaciones —como producto–categoría, producto–marca o producto–motivo de devolución— se modelan explícitamente en la base de grafos.

En este pipeline, la consulta se traduce a operaciones que permiten recuperar nodos y relaciones relevantes, evitando joins complejos o lógica relacional dispersa. El resultado consiste típicamente en conjuntos de entidades relacionadas o en patrones relacionales que no podrían expresarse de manera natural mediante una base tabular o vectorial.

Los resultados del grafo pueden utilizarse directamente como respuesta o bien combinarse con otros módulos, por ejemplo filtrando resultados tabulares a partir de identificadores obtenidos desde el grafo.

2.6 Integración final y generación de la respuesta

Una vez obtenidos los resultados desde el pipeline correspondiente, el sistema integra la información para construir la respuesta final.

En los casos vectoriales, el LLM cumple un rol central como generador de la respuesta, utilizando los fragmentos recuperados como contexto. En las consultas tabulares y de grafo, el modelo puede utilizarse como capa de presentación, transformando resultados estructurados en una respuesta conversacional clara.

Este diseño permite que el modelo de lenguaje actúe como **componente de síntesis y comunicación**, mientras que la obtención de la información se mantiene controlada por las bases especializadas.

2.7 Consideraciones finales sobre la ejecución

La ejecución del sistema demuestra que un enfoque híbrido, basado en múltiples tipos de bases de datos coordinadas por un clasificador de intención, resulta adecuado para escenarios reales de atención al cliente.

Cada tipo de consulta se resuelve utilizando la herramienta más apropiada, evitando forzar soluciones genéricas y mejorando tanto la precisión como la interpretabilidad del sistema.
