



FACULTAD DE
CIENCIAS EXACTAS,
INGENIERIA Y AGRIMENSURA



Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento del Lenguaje Natural

“Trabajo Práctico Final”

Autora: Valentina Balverdi - (B-6588/9)

Profesores:

- Juan Pablo Manson
- Alan Geray
- Constantino Ferrucci

Fecha de entrega: 7 de diciembre

2025

Introducción

El presente trabajo práctico tiene como objetivo desarrollar un asistente virtual capaz de responder preguntas relacionadas con productos de una empresa de electrodomésticos utilizando técnicas de Procesamiento del Lenguaje Natural. Para ello, se integran distintos tipos de datos y modelos con el fin de implementar un sistema de Recuperación con Generación Aumentada (RAG) y, posteriormente, un agente autónomo basado en el paradigma ReAct.

El trabajo se divide en dos ejercicios principales.

En el Ejercicio 1, se construye un sistema RAG que combina tres fuentes independientes de información:

- una base vectorial con documentos textuales,
- una base tabular con datos estructurados, y
- una base de grafos que modela relaciones entre productos.

Además, se implementa un clasificador de intención para determinar la fuente adecuada para cada consulta y un pipeline de recuperación que combina búsqueda semántica e instrucciones generadas por un modelo de lenguaje.

En el Ejercicio 2, el sistema se amplía mediante la creación de un agente autónomo capaz de seleccionar herramientas especializadas y ejecutar consultas más complejas. Dicho agente utiliza LangChain y el enfoque ReAct para razonar, elegir acciones y combinar información de distintas fuentes de manera dinámica.

Este informe describe los objetivos, la metodología, el desarrollo y los resultados obtenidos en ambos ejercicios, junto con las conclusiones generales del trabajo.

Ejercicio 1 – RAG: Sistema de Recuperación con Generación Aumentada

Objetivo del Ejercicio 1

El objetivo del Ejercicio 1 es desarrollar un sistema RAG capaz de responder consultas sobre productos de una empresa de electrodomésticos utilizando tres fuentes de datos complementarias:

- Vectorial: documentos textuales procesados mediante embeddings.
- Tabular: información estructurada en DataFrames.
- Grafo: relaciones entre productos representadas mediante nodos y aristas.

El sistema debe seleccionar de manera adecuada la fuente correspondiente a cada consulta y devolver una respuesta basada en la información recuperada. Para ello, se requiere:

1. construir las tres bases de datos,
2. implementar un clasificador de intención que determine qué fuente utilizar,
3. diseñar un pipeline de recuperación híbrido,
4. e integrar la recuperación con un modelo de lenguaje generativo.

Metodología

Para el desarrollo del Ejercicio 1 se diseñó un sistema de Recuperación con Generación Aumentada (RAG) basado en la integración de múltiples fuentes de datos heterogéneas, con el objetivo de responder consultas en lenguaje natural de manera precisa y fundamentada en evidencia.

Fuentes de datos

Se utilizaron los conjuntos de datos provistos en el enunciado del trabajo práctico, organizados según su naturaleza:

- Datos no estructurados, compuestos por documentos textuales como manuales de productos, reseñas de usuarios y preguntas frecuentes (FAQs).
- Datos estructurados, almacenados en tablas con información de productos, precios, stock, ventas, devoluciones y vendedores.
- Datos relacionales, representados mediante relaciones entre productos, categorías y preguntas frecuentes, utilizados para la construcción de una base de grafos.

Métodos y técnicas utilizadas

El enfoque metodológico se basa en la separación clara entre las etapas de recuperación y generación:

- La recuperación de información se realiza mediante mecanismos especializados según el tipo de fuente (semántica para texto, consultas dinámicas para tablas y consultas relacionales para el grafo).
- La generación de respuestas se delega a un modelo de lenguaje, que actúa exclusivamente como sintetizador de la información recuperada.
- Se implementó un clasificador de intención para determinar automáticamente qué fuente de datos resulta más adecuada para cada consulta del usuario.

Este diseño permite limitar el uso del modelo de lenguaje, reducir alucinaciones y garantizar que las respuestas estén basadas en datos verificables.

Herramientas y tecnologías empleadas

El sistema fue desarrollado íntegramente en un entorno de Google Colab, utilizando las siguientes tecnologías principales:

- pandas para la manipulación de datos estructurados.
- ChromaDB como base de datos vectorial.
- Neo4j para el almacenamiento y consulta de la base de grafos.
- scikit-learn para la implementación del clasificador de intención entrenado.
- LangChain para la integración con modelos de lenguaje y la generación dinámica de consultas.
- Modelos de lenguaje alojados en la nube (Gemini) para clasificación, planificación de consultas y generación de respuestas.

Pasos seguidos

Los pasos generales seguidos para cumplir los objetivos del ejercicio fueron:

1. Análisis y organización de las fuentes de datos según su estructura.
2. Construcción de las tres bases de datos (vectorial, tabular y de grafos).
3. Implementación de un clasificador de intención para la selección de la fuente adecuada.
4. Diseño de un pipeline de recuperación que permita consultar dinámicamente cada fuente.
5. Integración del sistema de recuperación con un modelo de lenguaje generativo para producir las respuestas finales.

Desarrollo

En esta sección se describen los pasos de implementación del sistema RAG correspondiente al Ejercicio 1, detallando la construcción de las tres fuentes de información, el clasificador de intención y los mecanismos de recuperación e integración del sistema.

Implementación de la base vectorial

La base vectorial se diseñó para resolver consultas relacionadas con información no estructurada, tales como manuales de productos, reseñas de usuarios y preguntas frecuentes (FAQs). Este tipo de consultas requiere capturar similitud semántica más allá de coincidencias léxicas, por lo que se adoptó un enfoque basado en embeddings.

Los documentos textuales fueron previamente preprocesados y almacenados en un formato unificado. El análisis exploratorio mostró que la longitud máxima de los textos no superaba los 1000 caracteres, por lo que no fue necesario aplicar una fragmentación adicional mediante text splitting. No obstante, se definió conceptualmente un tamaño de fragmento máximo con el objetivo de asegurar la escalabilidad del sistema ante documentos de mayor extensión.

Cada documento fue transformado en una representación vectorial utilizando el modelo de embeddings paraphrase-multilingual-mppnet-base-v2, seleccionado por su buen desempeño en idioma español y su capacidad para capturar similitud semántica entre textos parafraseados. Este enfoque permite recuperar documentos relevantes incluso cuando la consulta del usuario no coincide literalmente con el contenido almacenado.

Los vectores generados se almacenaron en una base vectorial implementada con ChromaDB, que permite realizar búsquedas por similitud utilizando métricas de distancia en el espacio vectorial. Para interactuar con esta fuente se desarrolló una función de consulta que, dada una pregunta del usuario y un valor k, recupera los fragmentos más cercanos según su proximidad semántica.

Implementación de la fuente tabular

La fuente tabular se diseñó para responder consultas que involucran información estructurada, tales como precios de productos, stock disponible, ventas, devoluciones y datos de vendedores. Este tipo de consultas requiere operaciones de filtrado, agregación y comparación sobre tablas, por lo que se trabajó directamente con DataFrames de pandas.

Los datos estructurados fueron cargados desde los archivos provistos en el enunciado y organizados en distintos DataFrames según su naturaleza (productos, ventas, inventario, devoluciones, entre otros). Para evitar incluir tablas completas dentro del contexto del modelo de lenguaje se adoptó una estrategia basada en metadatos.

A partir de cada DataFrame se extrajo información relevante como:

- nombres de columnas,
- valores únicos para variables categóricas,
- rangos, mínimos y máximos para variables numéricas.
-

Estos metadatos se utilizaron para construir un prompt especializado, en el cual el modelo de lenguaje actúa como un “analista de datos” encargado de interpretar la consulta del usuario y generar un plan de filtrado en formato estructurado. Dicho plan especifica la tabla a consultar, las columnas involucradas y las condiciones a aplicar.

Una vez generado el plan, este es validado y ejecutado dinámicamente sobre los DataFrames mediante código en pandas. El resultado se devuelve como una tabla filtrada, que luego puede ser utilizada como evidencia para la generación de la respuesta final.

Implementación de la base de grafos

La base de grafos se incorporó para modelar relaciones estructurales entre entidades, las cuales resultan poco naturales de representar mediante tablas y difíciles de recuperar mediante búsqueda semántica. En particular, se buscó capturar relaciones como la pertenencia de un producto a una categoría, la compatibilidad entre productos y la asociación entre productos y sus preguntas frecuentes (FAQs).

Para ello, se utilizó Neo4j como sistema gestor de base de datos de grafos, aprovechando su modelo basado en nodos y relaciones y su lenguaje de consulta Cypher.

A partir de los datos estructurados y documentales, se prepararon DataFrames intermedios que representan explícitamente nodos y relaciones

Los productos se modelaron como nodos principales del grafo, identificados por un identificador único. Las categorías se representaron como nodos independientes, permitiendo agrupar productos bajo una misma clasificación comercial. Las FAQs se incorporaron como nodos asociados a cada producto, facilitando consultas que requieren navegar desde un producto hacia su información técnica frecuente.

Las relaciones se insertaron en la base mediante operaciones idempotentes (MERGE), garantizando la integridad del grafo y evitando duplicaciones en ejecuciones sucesivas.

Asimismo, se definieron constraints de unicidad sobre los identificadores clave de cada tipo de nodo.

Para permitir consultas dinámicas, se implementó una interfaz que utiliza un modelo de lenguaje para traducir preguntas en lenguaje natural a consultas Cypher de solo lectura. El modelo recibe únicamente el esquema del grafo y un conjunto estricto de reglas, de modo que las consultas generadas no puedan modificar el estado de la base ni ejecutar operaciones peligrosas.

Este componente permite responder consultas que requieren navegación entre entidades, como:

- productos pertenecientes a una categoría,
- compatibilidades entre productos,
- preguntas frecuentes asociadas a un producto específico.

La base de grafos complementa así a las fuentes vectorial y tabular, ampliando la capacidad del sistema para responder consultas relacionales de forma eficiente y estructurada.

Modelo de clasificación de intención

Para dirigir cada consulta hacia la fuente de datos correspondiente, se implementaron dos clasificadores de intención: uno entrenado y otro basado en un modelo de lenguaje.

El primero consiste en un clasificador supervisado entrenado a partir de un conjunto sintético de consultas representativas, diseñado para cubrir preguntas relacionadas con uso de productos, información estructurada y relaciones entre entidades. El modelo utiliza un esquema de vectorización TF-IDF con unigramas y bigramas, seguido de un clasificador lineal. Dado el tamaño reducido del conjunto de entrenamiento, la evaluación se realizó de manera exploratoria, observándose un desempeño adecuado para distinguir entre los distintos tipos de consulta, con errores concentrados principalmente en casos ambiguos.

El segundo enfoque se basa en un modelo de lenguaje grande utilizando few-shot prompting. En este caso, el LLM recibe una breve descripción de cada intención junto con ejemplos etiquetados y devuelve directamente la clase correspondiente. Este método demostró una mayor capacidad de generalización semántica y obtuvo un desempeño perfecto sobre el conjunto de evaluación, a costa de una mayor latencia y dependencia de una API externa.

Ambos clasificadores fueron integrados al sistema final, permitiendo seleccionar cuál utilizar mediante configuración. En la implementación principal se prioriza el clasificador basado en LLM debido a su mayor robustez ante variaciones lingüísticas, mientras que el modelo entrenado queda disponible como alternativa reproducible y de bajo costo computacional.

Búsqueda híbrida y re-ranking

Para la recuperación de información a partir de documentos no estructurados, se implementó un pipeline de búsqueda híbrida y multi-etapa, cuyo objetivo es maximizar la

relevancia de los fragmentos recuperados y reducir la probabilidad de respuestas incorrectas o poco específicas.

En una primera etapa, el sistema combina búsqueda semántica, basada en embeddings almacenados en una base vectorial, con búsqueda por palabras clave mediante el algoritmo BM25. Esta combinación permite capturar tanto similitud conceptual como coincidencias léxicas exactas, relevantes en consultas técnicas o específicas. Los resultados de ambas estrategias se integran utilizando Reciprocal Rank Fusion (RRF), un método robusto que favorece los documentos bien posicionados en uno o ambos rankings sin depender de la escala de los puntajes individuales.

Con el fin de mejorar la precisión en la selección de la fuente de información, el pipeline incorpora además una etapa de priorización dinámica por tipo de documento. Dado que los manuales, las FAQs y las reseñas cumplen roles distintos dentro del sistema, se analizan patrones léxicos presentes en la consulta del usuario (como verbos de instrucción, unidades técnicas u opiniones) para estimar qué tipo de fuente resulta más adecuada. A partir de este análisis se asignan pesos relativos a cada tipo de documento, y los pools de resultados se combinan mediante una variante ponderada de RRF, favoreciendo aquellos fragmentos provenientes de la fuente más pertinente para la consulta planteada.

Finalmente, sobre el conjunto reducido de candidatos obtenidos tras la fusión híbrida y la priorización por fuente, se aplica un re-ranking final mediante un modelo Cross-Encoder multilingüe. Este modelo evalúa directamente pares (consulta, fragmento) y permite refinar el orden de los resultados en función de su relevancia contextual.

Elección del LLM y entorno de ejecución

Para las tareas de interpretación de consultas, generación de planes estructurados y síntesis de respuestas, se utilizó un modelo de lenguaje alojado en la nube, accesible mediante API desde el entorno de Google Colab. Esta decisión permite garantizar reproducibilidad, simplicidad de ejecución y evitar dependencias de hardware local, como GPUs o grandes requerimientos de memoria.

El modelo seleccionado fue Gemini 2.0 Flash, elegido por su buen desempeño en comprensión y generación de texto en español, su capacidad para seguir instrucciones estructuradas y su baja latencia. Estas características resultan adecuadas para un sistema RAG, donde el modelo no actúa como fuente primaria de conocimiento, sino como un componente encargado de interpretar consultas, generar instrucciones formales y redactar respuestas basadas exclusivamente en evidencia recuperada.

Integración para la generación y conversación

La integración final del sistema se realiza mediante un flujo conversacional que coordina la clasificación de intención, la recuperación de información y la generación de respuestas en lenguaje natural. Este flujo permite al asistente responder de forma dinámica y contextualizada a consultas encadenadas de los usuarios.

Ante cada consulta, el sistema determina la intención predominante y ejecuta el módulo de recuperación correspondiente. Los resultados obtenidos se formatean como evidencia

textual o estructurada y se incorporan al prompt de generación. El modelo de lenguaje utiliza dicha evidencia como contexto para redactar la respuesta final, sin acceso directo a las fuentes originales, lo que permite controlar el origen de la información utilizada..

El asistente incorpora una memoria conversacional de corto plazo, donde se almacenan los últimos turnos del diálogo. Esta memoria se utiliza para resolver referencias implícitas, como pronombres o consultas anafóricas, permitiendo interpretar correctamente preguntas dependientes del contexto previo. La memoria se emplea únicamente para desambiguación y nunca como fuente de información factual.

Asimismo, el sistema contempla mecanismos de fallback. Si una consulta no puede resolverse adecuadamente con la fuente inicialmente seleccionada o no se recupera evidencia relevante, el asistente intenta una recuperación alternativa o solicita al usuario que reformule la pregunta, indicando la falta de información disponible. Este comportamiento contribuye a mejorar la robustez del sistema y evita respuestas especulativas.

Resultados

En esta sección se presentan los resultados obtenidos a partir de la puesta en funcionamiento del sistema RAG desarrollado en el Ejercicio 1. El objetivo principal de las pruebas fue verificar que el asistente sea capaz de interpretar correctamente las consultas del usuario, seleccionar la fuente de información adecuada y generar respuestas coherentes basadas en la evidencia recuperada.

Para la evaluación se realizaron consultas representativas de cada tipo de fuente contemplada en el sistema:

- consultas semánticas sobre uso, mantenimiento y características generales de productos, resueltas mediante la base vectorial;
- consultas estructuradas sobre precios, stock, ventas y devoluciones, resueltas mediante la fuente tabular;
- consultas relacionales sobre categorías, compatibilidades y preguntas frecuentes, resueltas mediante la base de grafos.

Los resultados muestran que el sistema logra direccionar correctamente la mayoría de las consultas hacia la fuente correspondiente, recuperando información relevante y generando respuestas consistentes con los datos disponibles. En los casos en los que la consulta presenta ambigüedad o no existe información suficiente en las fuentes, el asistente evita generar respuestas especulativas y solicita al usuario reformular la pregunta, cumpliendo con los criterios de robustez definidos para el sistema.

Asimismo, las pruebas evidencian el correcto funcionamiento del flujo de integración entre clasificación de intención, recuperación de información y generación de respuestas, así como el uso de la memoria conversacional para interpretar consultas encadenadas y referencias implícitas a turnos anteriores.

Se presentan ejemplos de interacción con el asistente, donde se observan consultas de distinto tipo y las respuestas generadas a partir de cada una de las fuentes de información, junto con la trazabilidad interna del sistema.

 **Usuario 1:** ¿Cómo uso mi licuadora para hacer smoothies?

 **Asistente 1:** Para hacer un smoothie con tu Compacto Licuadora, seguí estos pasos:

1. Lavar y cortar las frutas en trozos medianos (2-3 cm).
 2. Colocar los ingredientes líquidos primero (leche, yogur, jugo).
 3. Agregar las frutas y hielo en la jarra.
 4. Cerrar la tapa asegurándose del cierre hermético.
 5. Comenzar en velocidad baja (nivel 1-2) por 10 segundos.
 6. Aumentar gradualmente a velocidad alta (nivel 4-5).
 7. Procesar durante 45-60 segundos hasta textura homogénea.
 8. Usar función PULSE para romper trozos grandes si es necesario.
 9. Verificar consistencia y procesar 10-15 seg adicionales si hace falta.
 10. Apagar y desconectar antes de retirar la jarra.
 11. Servir inmediatamente para mejor sabor y textura.
-  **Trazabilidad:** {'intencion': 'vectorial', 'fuente': 'VECTORIAL', 'memoria_expandida_para_retrieval': False, 'fallback_vectorial': False}
-

 **Usuario 2:** ¿y cómo la limpio?

 **Asistente 2:** Para limpiar tu Compacto Licuadora, seguí estos pasos:

1. Desconéctala primero.
 2. Las piezas removibles pueden lavarse con agua tibia y jabón.
 3. La base debe limpiarse solo con un paño húmedo.
 4. NO sumergir la base en agua.
-  **Trazabilidad:** {'intencion': 'vectorial', 'fuente': 'VECTORIAL', 'memoria_expandida_para_retrieval': False, 'fallback_vectorial': False}
-

 **Usuario 3:** ¿Qué problemas tiene el Ventilador de Torre 2024 (P0199)?

 **Asistente 3:** Según la evidencia, los problemas comunes del Ventilador de Torre 2024 son:

- * No funciona correctamente.
- * Muestra un código de error.
- * Se rompe a los pocos días de uso.
- * Es complicado y ruidoso.

Para más detalles sobre los problemas y sus soluciones, revisá el manual del producto (código P0199). Si tenés dudas, contactá al servicio de atención al cliente o al servicio técnico oficial. El ventilador tiene 18 meses de garantía oficial que cubre defectos de fábrica y fallas en componentes eléctricos.

⌚ Trazabilidad: {'intencion': 'vectorial', 'fuente': 'VECTORIAL',
'memoria_expandida_para_retrieval': False, 'fallback_vectorial': False}
=====

👤 Usuario 4: ¿Necesito algún accesorio adicional en la Deluxe Procesadora?

🤖 Asistente 4: El Deluxe Procesadora de TechHome está diseñado para uso doméstico. Revisá el manual del producto (código P0010) para más detalles. Ante cualquier duda, contactá a nuestro servicio de atención al cliente.

⌚ Trazabilidad: {'intencion': 'grafo', 'fuente': 'VECTORIAL',
'memoria_expandida_para_retrieval': False, 'fallback_vectorial': True}
=====

👤 Usuario 5: Reseñas de Aire Portátil Pro

🤖 Asistente 5: La evidencia que tengo son preguntas frecuentes (FAQs), no reseñas de usuarios.

Los FAQs del Aire Portátil Pro ([1],[2],[3],[4]) indican:

- * Funciona con 220V.
- * El consumo es de 1300W.
- * Recomiendan usar un estabilizador de tensión.
- * Tiene 18 meses de garantía oficial que cubre defectos de fábrica y fallas en componentes eléctricos. Para hacerla válida, contactá al servicio técnico oficial.

También hay información sobre el Advanced Aire Portátil Pro [5]:

- * Tiene 18 meses de garantía oficial que cubre defectos de fábrica y fallas en componentes eléctricos. Para hacerla válida, contactá al servicio técnico oficial.

¿Querías saber algo más sobre el Aire Portátil Pro o el Advanced Aire Portátil Pro?

⌚ Trazabilidad: {'intencion': 'vectorial', 'fuente': 'VECTORIAL',
'memoria_expandida_para_retrieval': False, 'fallback_vectorial': False}
=====

👤 Usuario 6: ¿Cuáles son las licuadoras de menos de 2000 dólares?

🤖 Asistente 6: Las licuadoras de menos de 2000 dólares son:

- * Licuadora (P0001): \$283.63
- * Licuadora (P0002): \$1273.06
- * Plus Licuadora Pro (P0003): \$329.07
- * Compacto Licuadora (P0004): \$259.42
- * Licuadora (P0006): \$1837.15
- * Ultra Licuadora (P0008): \$1201.99

⌚ Trazabilidad: {'intencion': 'tabular', 'fuente': 'TABULAR',
'memoria_expandida_para_retrieval': False, 'fallback_vectorial': False}

=====

👤 Usuario 7: ¿y las licuadoras de menos de 500?

🤖 Asistente 7: Las licuadoras de menos de 500 dólares son:

- * Licuadora (P0001): \$283.63
- * Plus Licuadora Pro (P0003): \$329.07
- * Compacto Licuadora (P0004): \$259.42

🧭 Trazabilidad: {'intencion': 'tabular', 'fuente': 'TABULAR', 'memoria_expandida_para_retrieval': False, 'fallback_vectorial': False}

=====

👤 Usuario 8: ¿Qué productos están relacionados con la categoría Cocina?

🤖 Asistente 8: Los siguientes productos están relacionados con la categoría Cocina:

- * Licuadora (P0001)
- * Licuadora (P0002)
- * Plus Licuadora Pro (P0003)
- * Compacto Licuadora (P0004)
- * Licuadora (P0005)
- * Licuadora (P0006)
- * Licuadora (P0007)
- * Ultra Licuadora (P0008)
- * Procesadora (P0009)
- * Deluxe Procesadora (P0010)
- * Profesional Procesadora 2024 (P0011)
- * Advanced Procesadora II (P0012)
- * Procesadora (P0013)
- * Premium Picadora (P0014)
- * Picadora (P0015)
- * Super Picadora (P0016)
- * Profesional Batidora de Mano (P0017)
- * Batidora de Mano II (P0018)
- * Batidora de Mano (P0019)
- * Batidora de Pie (P0020)

🧭 Trazabilidad: {'intencion': 'grafo', 'fuente': 'GRAFO', 'memoria_expandida_para_retrieval': False, 'fallback_vectorial': False}

=====

👤 Usuario 9: ¿y dentro de esa categoría, cuáles son licuadoras?

🤖 Asistente 9: Dentro de la categoría Cocina, y según la evidencia actual, las licuadoras son:

- * Licuadora. La evidencia ([1],[2],[3]) indica que hay modelos que funcionan con 110-220V y otros con 12V. El consumo también varía: 1350W o 650W. Se recomienda usar un estabilizador de tensión.
- * Compacto Licuadora (P0004): Tiene una potencia de 1000W, funciona con 220V y pesa 48.7 kg. Tiene 24 meses de garantía.

🧭 Trazabilidad: {'intencion': 'grafo', 'fuente': 'VECTORIAL', 'memoria_expandida_para_retrieval': True, 'fallback_vectorial': True}

=====

👤 Usuario 10: ¿Qué productos compatibles tiene la max plancha a vapor?
🤖 Asistente 10: No encontré información sobre productos compatibles con la Max Plancha a Vapor. Sin embargo, encontré productos compatibles con otros productos:

- * **Pro Tender Eléctrico** (`P0271`): Comparte jarra con la Planchita de Pelo.
- * **Planchita de Pelo** (`P0298`): Comparte accesorios con la Planchita de Pelo.
- * **Digital Plancha a Vapor Plus** (`P0283`): Comparte jarra con la Planchita de Pelo.
- * **Smart Secarropa** (`P0256`): Comparte panel de control con la Planchita de Pelo.
- * **Lavarropas Carga Superior** (`P0237`): Comparte panel de control con la Planchita de Pelo.
- * **Plancha Seca** (`P0287`): Comparte jarra con Lavarropas Semiautomático X.
- * **Max Lavavajillas** (`P0273`): Comparte panel de control con Lavarropas Semiautomático X.
- * **Centro de Planchado** (`P0291`): Comparte accesorios con Lavarropas Semiautomático X.
- * **Secarropa** (`P0261`): Comparte accesorios con Lavarropas Semiautomático X.
- * **Lavavajillas Compacto** (`P0277`): Comparte motor con Lavarropas Semiautomático X.

¿Te interesaría que busque productos compatibles con alguno de estos productos?

🧭 Trazabilidad: {'intencion': 'grafo', 'fuente': 'VECTORIAL', 'memoria_expandida_para_retrieval': False, 'fallback_vectorial': True}

=====

👤 Usuario 11: ¿Cuál es el voltaje de la Cafetera Galáctica 9000?
🤖 Asistente 11: No encontré información sobre la "Cafetera Galáctica 9000". Sin embargo, encontré información sobre los siguientes productos:

- * Deluxe Cafetera 2024: Funciona con 12V.
- * Super Picadora: Funciona con 220V.
- * Sandwichera 3000: Funciona con 12V.
- * Olla Arrocera 3000: Funciona con 220V.
- * Lavavajillas: Funciona con 12V.

¿Te referías a alguno de estos productos?

🧭 Trazabilidad: {'intencion': 'vectorial', 'fuente': 'VECTORIAL', 'memoria_expandida_para_retrieval': False, 'fallback_vectorial': False}

Ejercicio 2 – Evolución del RAG a un Agente Autónomo

Objetivo

El objetivo del Ejercicio 2 es transformar el sistema RAG desarrollado previamente en un agente autónomo capaz de decidir dinámicamente qué fuente de información utilizar para responder una consulta del usuario.

Para ello, se adopta el paradigma ReAct (Reason + Act), en el cual el modelo de lenguaje razona sobre la consulta, selecciona la herramienta más adecuada, ejecuta la acción correspondiente y utiliza la evidencia obtenida para generar la respuesta final.

El agente debe poder interactuar con documentos textuales, datos tabulares, un grafo de relaciones y consultas analíticas con generación de gráficos, respondiendo siempre en español, evitando información no respaldada por evidencia y solicitando reformulación cuando los datos disponibles no resultan suficientes.

Metodología

En primer lugar, se reutilizaron las fuentes de información ya construidas en el Ejercicio 1 (documental, tabular y de grafos), encapsulando la lógica de acceso a cada una dentro de herramientas independientes. A estas se añadió una herramienta adicional de analytics, orientada a consultas analíticas y generación de visualizaciones a partir de datos estructurados.

El agente fue diseñado para operar bajo el paradigma ReAct, donde el modelo de lenguaje analiza la consulta, decide qué herramienta utilizar, observa el resultado obtenido y, si es necesario, encadena nuevas acciones antes de producir una respuesta final. Este enfoque permite resolver consultas que requieren distintos tipos de evidencia sin depender de reglas rígidas o flujos predefinidos.

Para garantizar trazabilidad y robustez, se establecieron reglas estrictas en el prompt del sistema que obligan al agente a:

- utilizar herramientas antes de responder cuando se requiere evidencia,
- no inventar datos,
- y solicitar reformulación cuando la información disponible resulta insuficiente.

El desarrollo se realizó íntegramente en Google Colab, reutilizando el entorno y los datos del ejercicio anterior.

Entorno y herramientas utilizadas

El agente autónomo fue implementado utilizando las siguientes tecnologías:

- LangChain y LangGraph para la construcción del agente ReAct y la orquestación del ciclo Thought–Action–Observation.

- SQLite como base de datos local para consultas analíticas.
- SQLDatabase y QuerySQLDatabaseTool (LangChain Community) para la ejecución de consultas SQL desde el agente.
- matplotlib para la generación de gráficos a partir de resultados analíticos..
- Modelo de lenguaje Gemini (en la nube) como motor de razonamiento y decisión del agente.

Desarrollo

Encapsulamiento de fuentes de información como herramientas

En el Ejercicio 2, el sistema RAG desarrollado previamente se extiende hacia un esquema de agente autónomo, en el cual las distintas fuentes de información dejan de ser invocadas de manera fija y pasan a estar disponibles como herramientas independientes que el modelo puede seleccionar dinámicamente según la consulta del usuario.

Cada una de las fuentes implementadas en el Ejercicio 1 se encapsuló como una herramienta con una interfaz uniforme, de modo que el agente pueda interactuar con ellas sin conocer los detalles internos de su implementación. Este diseño permite separar claramente la lógica de recuperación de la lógica de decisión, y constituye la base del comportamiento autónomo del sistema.

La herramienta de búsqueda documental (doc_search) encapsula el pipeline de recuperación híbrida sobre documentos no estructurados. Internamente, reutiliza la búsqueda semántica basada en embeddings, la búsqueda léxica mediante BM25, la fusión de resultados y el re-ranking final. Esta herramienta se utiliza para responder consultas relacionadas con uso de productos, instrucciones, mantenimiento, especificaciones técnicas y opiniones de usuarios. La salida consiste en fragmentos textuales relevantes acompañados de metadatos, que luego son utilizados como evidencia para la generación de la respuesta final.

La herramienta de consulta tabular (table_search) encapsula el mecanismo de generación y ejecución de planes de consulta sobre datos estructurados. A partir de una consulta en lenguaje natural, el modelo de lenguaje genera un plan en formato JSON que especifica la tabla a consultar, los filtros a aplicar y las columnas a devolver. Dicho plan es validado y ejecutado dinámicamente sobre DataFrames de pandas, y el resultado se devuelve como una tabla estructurada. Esta herramienta se emplea para responder preguntas relacionadas con precios, stock, ventas, devoluciones y tickets.

La herramienta de consulta a grafos (graph_search) reutiliza la base de datos Neo4j construida en el Ejercicio 1. Su función consiste en traducir consultas en lenguaje natural a consultas Cypher de solo lectura, respetando un conjunto estricto de reglas que garantizan la seguridad y la integridad del grafo. Esta herramienta permite responder consultas que requieren navegar relaciones entre entidades, como productos pertenecientes a una categoría, compatibilidades entre productos o preguntas frecuentes asociadas a un producto específico.

El encapsulamiento de estas tres fuentes como herramientas independientes permite que el agente seleccione, combine o reintepte consultas sobre distintas fuentes sin modificar la lógica de cada recuperador.

Herramienta de analítica y visualización

Además de las fuentes utilizadas en el Ejercicio 1, en el Ejercicio 2 se incorporó una herramienta de analítica destinada a responder consultas que requieren agregaciones, distribuciones y visualizaciones gráficas sobre los datos estructurados. Esta herramienta amplía las capacidades del asistente, permitiendo cubrir un tipo de consultas que no se resuelven adecuadamente mediante simples filtros tabulares.

Para su implementación se construyó una base de datos SQLite a partir de los DataFrames existentes (ventas, productos, inventario y devoluciones). Esta base se genera dinámicamente al iniciar el sistema, garantizando consistencia entre los datos utilizados por las distintas herramientas. La conexión y ejecución de consultas SQL se realiza mediante integraciones provistas por LangChain, utilizando SQLDatabase y QuerySQLDatabaseTool.

Ante una consulta del usuario, la herramienta genera dinámicamente una consulta SQL a partir del lenguaje natural, utilizando un modelo de lenguaje como traductor entre la pregunta y el esquema de la base. El modelo recibe únicamente el esquema de las tablas disponibles y un conjunto de reglas estrictas que limitan las operaciones permitidas a consultas de solo lectura. La consulta generada es limpiada, validada y ejecutada sobre la base SQLite, y el resultado se transforma en una tabla estructurada.

Adicionalmente, la herramienta incorpora una detección automática de solicitudes de visualización. Cuando la consulta incluye términos asociados a gráficos o distribuciones, los resultados de la consulta SQL se utilizan para generar una visualización mediante matplotlib. El gráfico resultante se guarda como un archivo PNG y su ruta se devuelve explícitamente como parte de la evidencia, permitiendo su visualización dentro del entorno de ejecución.

Esta herramienta permite responder consultas analíticas complejas, como distribuciones por categoría o comparaciones agregadas, integrando generación de consultas, ejecución y visualización dentro de un único componente controlado.

Definición de herramientas y reglas del agente

Para transformar el sistema RAG en un agente autónomo, cada fuente de información se encapsuló como una herramienta independiente, siguiendo el modelo propuesto por LangChain. Cada herramienta expone una interfaz clara y devuelve evidencia estructurada que el agente utiliza para construir la respuesta final.

Se definieron cuatro herramientas principales:

- doc_search: recupera fragmentos relevantes de documentos no estructurados (manuales, FAQs y reseñas) utilizando el pipeline de búsqueda híbrida con re-ranking.

- `table_search`: ejecuta consultas dinámicas sobre DataFrames, generadas por un modelo de lenguaje en forma de planes estructurados.
- `graph_search`: traduce consultas en lenguaje natural a Cypher y las ejecuta sobre la base de grafos Neo4j para resolver relaciones entre entidades.
- `analytics_tool`: genera y ejecuta consultas SQL sobre SQLite y, cuando corresponde, produce visualizaciones gráficas.

El comportamiento del agente se controla mediante un *system prompt* con reglas explícitas que obligan al uso de herramientas para obtener evidencia antes de responder, restringen la invención de datos y priorizan la herramienta de analítica cuando la consulta solicita gráficos o distribuciones. Estas reglas garantizan que las respuestas se basen exclusivamente en información verificada y que la selección de herramientas sea coherente con la intención de la consulta.

Integración del agente ReAct y flujo de ejecución

La integración final del sistema se realizó mediante un agente autónomo basado en el paradigma ReAct (Reason + Act), que permite al modelo de lenguaje decidir de forma dinámica qué herramienta utilizar en función de la consulta del usuario.

El agente fue implementado utilizando LangGraph, que orquesta el ciclo de razonamiento y ejecución de acciones, junto con LangChain, que encapsula cada fuente de datos como una herramienta reutilizable. El modelo de lenguaje actúa como componente de control, analizando la consulta, seleccionando la herramienta adecuada y utilizando la evidencia obtenida para construir la respuesta final.

Ante cada consulta, el flujo de ejecución es el siguiente:

1. El agente analiza la pregunta del usuario.
2. Selecciona una o más herramientas según la naturaleza de la consulta.
3. Ejecuta la herramienta elegida y recibe la evidencia correspondiente.
4. Genera la respuesta final utilizando exclusivamente dicha evidencia.

En el caso de consultas analíticas o de visualización, el agente prioriza el uso de la herramienta de `analytics`, que devuelve tanto los resultados tabulares como la ruta del gráfico generado. Para consultas textuales, estructuradas o relacionales, el agente utiliza las herramientas documentales, tabulares o de grafos respectivamente.

Este enfoque permite que el asistente no dependa de reglas rígidas ni de una clasificación previa de intención, sino que razone y actúe de manera autónoma, adaptando su comportamiento a cada consulta y combinando múltiples fuentes cuando resulta necesario.

Resultados

En este ejercicio se evaluó el funcionamiento del sistema una vez evolucionado hacia un agente autónomo basado en ReAct, con capacidad de seleccionar dinámicamente herramientas y combinar distintas fuentes de información según la consulta del usuario.

Las pruebas se realizaron mediante una sesión interactiva, planteando consultas representativas de los distintos tipos soportados por el agente: consultas documentales, tabulares, relacionales y analíticas con visualización.

Los resultados muestran que el agente es capaz de:

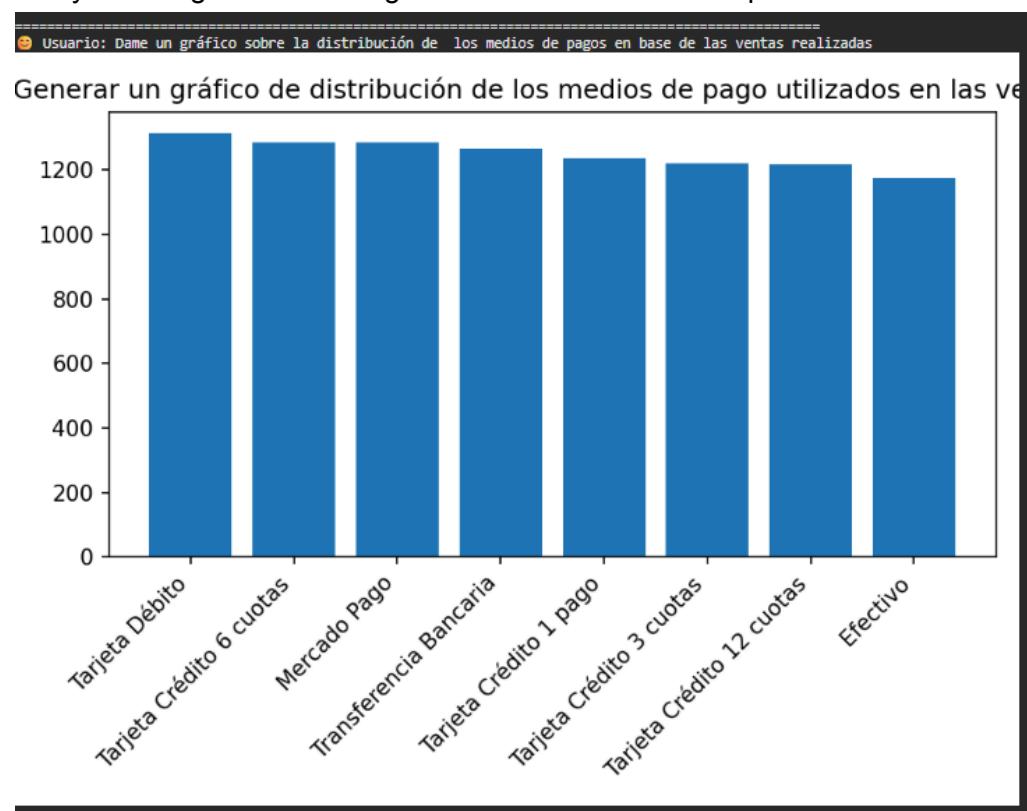
- Seleccionar correctamente la herramienta adecuada según el tipo de consulta.
- Ejecutar consultas sobre múltiples fuentes sin intervención manual.
- Generar respuestas basadas exclusivamente en la evidencia devuelta por las herramientas.
- Producir visualizaciones gráficas cuando la consulta lo requiere.

En particular, se verificó el correcto funcionamiento de la herramienta de analítica, donde el agente generó consultas SQL dinámicas y produjo gráficos de distribución a partir de los datos de ventas.

Cuando una consulta no pudo resolverse con la información disponible, el agente respondió solicitando una reformulación o mayor nivel de detalle, evitando respuestas especulativas. Este comportamiento confirma que las reglas impuestas al agente permiten mantener control sobre el origen de la información y refuerzan la confiabilidad del sistema.

Evidencia visual de funcionamiento

Se incluye una captura del entorno de ejecución mostrando consultas realizadas por el usuario, selección automática de herramientas por parte del agente y resultados obtenidos, incluyendo la generación de gráficos en los casos correspondientes.



😊 Usuario: ¿Cómo uso mi licuadora para hacer smoothies?

🤖 Asistente: Para preparar un smoothie de frutas en tu licuadora Compacto, te recomiendo seguir estos pasos:

1. Lavar y cortar las frutas en trozos medianos (2-3 cm).
 2. Colocar los ingredientes líquidos primero (leche, yogurt, jugo).
 3. Agregar las frutas y hielo en la jarra.
 4. Cerrar la tapa asegurándose del cierre hermético.
 5. Comenzar en velocidad baja (nivel 1-2) por 10 segundos.
 6. Aumentar gradualmente a velocidad alta (nivel 4-5).
 7. Procesar durante 45-60 segundos hasta obtener una textura homogénea.
 8. Usar la función PULSE para romper trozos grandes si es necesario.
 9. Verificar la consistencia y procesar 10-15 segundos adicionales si hace falta.
 10. Apagar y desconectar antes de retirar la jarra.
 11. Servir inmediatamente para un mejor sabor y textura.
-

😊 Usuario: ¿Cuáles son las licuadoras de menos de \$200?

🤖 Asistente: No encontré licuadoras con un precio menor a \$200. ¿Te gustaría que busque licuadoras en otro rango de precios?

😊 Usuario: ¿Cuáles son las licuadoras de menos de \$1000?

🤖 Asistente: Las licuadoras de menos de \$1000 son: Licuadora (P0001) por \$283.63, Plus Licuadora Pro (P0003) por \$329.07 y Compacto Licuadora (P0004) por \$259.42.

😊 Usuario: ¿Qué opinan los usuarios de la licuadora?

🤖 Asistente: Las opiniones de los usuarios sobre la licuadora son variadas. Algunos consideran que está bien por el precio, pero no esperan grandes resultados. Otros mencionan que tiene cosas buenas y malas. Además, la licuadora tiene una garantía de 6 a 18 meses que cubre defectos de fábrica y fallas en los componentes eléctricos.

😊 Usuario: ¿Qué voltaje requiere el rallador digital eléctrico?

🤖 Asistente: El Digital Rallador Eléctrico funciona con 220V y tiene un consumo de 1300W. Se recomienda usar un estabilizador de tensión. También existe el Eco Rallador Eléctrico que funciona con 110-220V y consume 1050W.

😊 Usuario: ¿Qué productos son compatibles con Max Plancha a Vapor?

 **Asistente:** Los productos compatibles con Max Plancha a Vapor son:
Lavarropas Semiautomático X (P0236), Compacto Centrifuga (P0267),
Lavavajillas Compacto (P0277), Centro de Planchado (P0291) y Centro de
Planchado (P0292).

=====

 **Usuario:** ¿Qué productos están relacionados con la categoría Cocina?

 **Asistente:** Los productos relacionados con la categoría Cocina son:
Licuadora (P0001, P0002, P0005, P0006, P0007), Plus Licuadora Pro
(P0003), Compacto Licuadora (P0004), Ultra Licuadora (P0008),
Procesadora (P0009, P0013), Deluxe Procesadora (P0010), Profesional
Procesadora 2024 (P0011), Advanced Procesadora II (P0012), Premium
Picadora (P0014), Picadora (P0015), Super Picadora (P0016), Profesional
Batidora de Mano (P0017), Batidora de Mano II (P0018), Batidora de Mano
(P0019), Batidora de Pie (P0020).

Propuestas de Mejora

Si bien el sistema desarrollado cumple con los objetivos planteados y demuestra un funcionamiento correcto tanto en el enfoque RAG como en su evolución hacia un agente autónomo, se identificaron algunas limitaciones que abren la posibilidad a mejoras futuras.

Una primera mejora posible se relaciona con el tratamiento de las reseñas de usuarios. Actualmente, las reseñas se procesan como documentos no estructurados dentro de la base vectorial, lo que permite su recuperación semántica pero no diferencia explícitamente entre opiniones positivas y negativas. Una mejora futura consistiría en incorporar técnicas de análisis de sentimiento o clasificación de opiniones, permitiendo identificar el tono general de una reseña y priorizar respuestas que reflejen mejor la percepción global de los usuarios sobre un producto.

Asimismo, podría fortalecerse la robustez del acceso a manuales y FAQs. Si bien el sistema recupera fragmentos relevantes mediante búsqueda híbrida y re-ranking, sería posible mejorar la identificación de secciones específicas y priorizar de forma más estricta.

En relación con el agente autónomo, una mejora importante consistiría en incorporar un mecanismo explícito de memoria conversacional. En la implementación actual, el agente procesa cada consulta de forma independiente, sin mantener estado entre turnos. Una extensión natural del sistema sería agregar una memoria estructurada que permita conservar información relevante del contexto, como el producto en foco o decisiones previas de selección de herramientas. Esto permitiría mejorar la coherencia en diálogos prolongados y habilitar comportamientos más avanzados en consultas encadenadas..

Conclusiones

En este trabajo práctico se desarrolló un asistente virtual para una empresa de electrodomésticos, integrando técnicas de Procesamiento del Lenguaje Natural, recuperación de información y modelos de lenguaje de gran escala. A lo largo de los dos ejercicios propuestos, se logró evolucionar progresivamente desde un sistema RAG clásico hacia un agente autónomo capaz de seleccionar herramientas y ejecutar acciones de manera dinámica.

En el Ejercicio 1, se implementó un sistema de Recuperación con Generación Aumentada que combina tres fuentes de información complementarias: una base vectorial para documentos no estructurados, una fuente tabular para datos estructurados y una base de grafos para modelar relaciones entre entidades. La incorporación de un clasificador de intención permitió direccionar cada consulta hacia la fuente más adecuada, mientras que el uso de búsqueda híbrida y re-ranking mejoró la calidad de los fragmentos recuperados. El enfoque adoptado demostró ser efectivo para responder consultas variadas, manteniendo un control estricto sobre el origen de la información utilizada en las respuestas.

En el Ejercicio 2, el sistema fue extendido hacia un agente autónomo basado en el paradigma ReAct. Esta evolución permitió eliminar la dependencia de una clasificación rígida de intención y habilitar un comportamiento más flexible, donde el modelo de lenguaje decide qué herramienta utilizar en función de la consulta. La incorporación de una herramienta de analítica con generación dinámica de consultas SQL y visualizaciones gráficas amplió significativamente el alcance del asistente, permitiendo responder consultas analíticas complejas que no podían resolverse únicamente mediante recuperación textual o tabular.

Los resultados obtenidos muestran que el agente es capaz de razonar sobre la consulta, utilizar herramientas especializadas para obtener evidencia y generar respuestas fundamentadas exclusivamente en información verificada. Asimismo, los mecanismos de fallback y solicitud de reformulación contribuyen a evitar respuestas especulativas, reforzando la confiabilidad del sistema.

En conjunto, el trabajo evidencia la importancia de separar claramente los procesos de recuperación y generación, así como el valor de combinar distintas representaciones de datos dentro de una arquitectura unificada. La evolución hacia un agente autónomo demuestra cómo los modelos de lenguaje pueden utilizarse no solo como generadores de texto, sino también como componentes de control capaces de coordinar múltiples fuentes de información de manera eficiente y segura.

Bibliografía

Fuentes de datos:
<https://drive.google.com/drive/folders/12FNaPe1Mqs3QQ8QBp78WUEXgTVKnVbeK?usp=sharing>

ChromaDB - Open-source Vector Database: <https://www.trychroma.com/>

Neo4j Cypher Query Language: <https://neo4j.com/docs/cypher-refcard/current/>

Gemini Model by Google: <https://ai.google.dev/api?hl=es-419&lang=python>