

# **TRABAJO PRÁCTICO FINAL - PROCESAMIENTO DEL LENGUAJE NATURAL**

**TUIA - Universidad Nacional de Rosario**

**Estudiante:** Sebastián Palacio **Dni:** 43491996

**Fecha de entrega:** 7 de diciembre de 2025

## Modelos utilizados

### Modelo de lenguaje: Groq – Llama-3.3-70b-versatile

Modelo de Lenguaje: Llama-3.3-70b-versatile (Groq API)

Un modelo de lenguaje es un sistema de inteligencia artificial capaz de interpretar, generar y razonar sobre texto natural. En este proyecto, el LLM cumple un rol central al encargarse de múltiples tareas clave del flujo de trabajo:

- Generar consultas SQL dinámicas para interactuar con la base de datos tabular.
- Generar consultas Cypher dinámicas para interactuar con la base de grafos Neo4j.
- Actuar como motor de razonamiento dentro del paradigma ReAct.
- Convertir la información recuperada mediante las herramientas en respuestas finales para el usuario.

Se seleccionó Llama-3.3-70b-versatile a través de la API de Groq debido a su excelente balance entre capacidad de razonamiento y velocidad de inferencia. Groq ofrece latencias extremadamente bajas gracias a su hardware especializado (LPU), lo cual es ideal para aplicaciones conversacionales donde la respuesta debe ser ágil. Además, el modelo soporta español nativamente y posee una ventana de contexto amplia que permite manejar prompts extensos con información de esquemas de bases de datos.

### Modelo de embedding:

#### paraphrase-multilingual-MiniLM-L12-v2

Un modelo de embedding es un modelo entrenado para convertir texto en representaciones numéricas (vectores) que capturan su significado semántico. En este trabajo, el modelo de embedding se utilizó para:

- Generar vectores de los documentos (FAQs, manuales, reseñas) almacenados en la base de datos vectorial.
- Convertir las consultas del usuario en embeddings para búsqueda semántica.
- Entrenar y utilizar el clasificador de intenciones basado en similitud.

Se seleccionó paraphrase-multilingual-MiniLM-L12-v2 por su combinación de eficiencia, velocidad y rendimiento semántico. Al ser un modelo optimizado y multilingüe, permite generar embeddings con bajo costo computacional mientras asegura una adecuada comprensión de consultas y documentos en español.

### Text Splitter:

Para el preprocesamiento de los documentos se utilizó una configuración de segmentación optimizada, con fragmentos de aproximadamente 500 caracteres y un solapamiento de 50 caracteres entre ellos. Esta estrategia permite preservar la coherencia semántica de cada fragmento y evitar la pérdida de información en los límites de segmentación, mejorando la calidad de las búsquedas posteriores en la base vectorial.

## Clasificador de intenciones:

La función del clasificador es identificar la intención de la consulta del usuario y determinar qué fuente de datos resulta más adecuada para resolverla, ya sea vectorial, tabular, de grafos o analítica. Se evaluaron dos enfoques: un clasificador basado en LLM Few-Shot con embeddings y un modelo de Regresión Logística entrenado.

Si bien ambos enfoques obtuvieron resultados similares en términos de accuracy, el método basado en LLM Few-Shot mostró mejores métricas de precisión y recall en las clases más complejas, particularmente en las consultas documentales. Esta capacidad, sumada a la posibilidad de incorporar nuevos ejemplos sin necesidad de reentrenar el modelo, justificó su adopción como clasificador principal del sistema.

## Arquitectura implementada:

El sistema implementa una arquitectura RAG multimodal que integra distintas fuentes de información. La base vectorial, construida con ChromaDB, contiene 3.050 documentos entre FAQs, manuales y reseñas de usuarios. La base tabular, implementada con Pandas y SQLite, almacena información de productos, ventas e inventario. Por su parte, la base de grafos, desarrollada en Neo4j, modela las relaciones entre productos, categorías, marcas y sucursales.

Estas fuentes se combinan de forma estratégica para responder consultas complejas, utilizando búsqueda semántica, consultas estructuradas y razonamiento relacional según el caso.

## Sistema dinámico y agente ReAct:

Uno de los principales logros del proyecto es la eliminación total del hardcodeo en las consultas a las bases de datos. El sistema genera dinámicamente consultas SQL y Cypher en tiempo real, adaptadas a cada pregunta formulada en lenguaje natural. Esto permite responder a una amplia variedad de consultas sin depender de patrones predefinidos.

Sobre esta arquitectura se implementó un agente autónomo basado en el paradigma ReAct, que coordina distintas herramientas de búsqueda y análisis. El agente incorpora memoria conversacional, lo que le permite mantener el contexto entre preguntas sucesivas y enriquecer automáticamente las consultas cuando existe continuidad temática.

## Resultados de ejecución:

### Ejercicio 1: Sistema RAG

El sistema RAG individual alcanzó una tasa de éxito del 83% completando exitosamente 5 de 6 pruebas. Las consultas sobre uso de productos ('¿Cómo uso mi licuadora para hacer

smoothies?') se resolvieron correctamente recuperando FAQs relevantes desde la base vectorial. Las búsquedas de productos con filtros ('¿Cuáles son las licuadoras de menos de 500 dólares?') funcionaron perfectamente identificando y listando productos específicos desde la base tabular.

Las consultas relacionales ('¿Qué productos están en la categoría Cocina?') se ejecutaron exitosamente retornando listas de productos organizadas por categorías desde la base de grafos. La única prueba parcialmente exitosa buscaba información técnica específica (voltaje del rallador eléctrico) que no existe en los datos sintéticos del trabajo práctico, pero el sistema funcionó correctamente clasificando y buscando en la fuente apropiada. Las consultas sobre marcas y relaciones ('¿Qué marcas tienen productos de climatización?') se completaron exitosamente identificando 5 marcas específicas del sector.

## **Ejercicio 2: Agente ReAct con Sistema Dinámico**

El agente ReAct alcanzó una tasa de éxito del 100% completando exitosamente las 10 pruebas programadas, incluyendo 6 consultas individuales y 4 pruebas de memoria conversacional. Las consultas individuales demostraron la capacidad del sistema para generar dinámicamente consultas SQL específicas como 'SELECT nombre, precio\_usd FROM productos WHERE nombre LIKE '%licuadora%' ORDER BY precio\_usd' para búsquedas de productos ordenadas por precio, y consultas Cypher como 'MATCH (p:Producto)-[:PERTENECE\_A]->(c:Categoria) WHERE c.name = "Cocina" RETURN p.name' para relaciones categóricas.

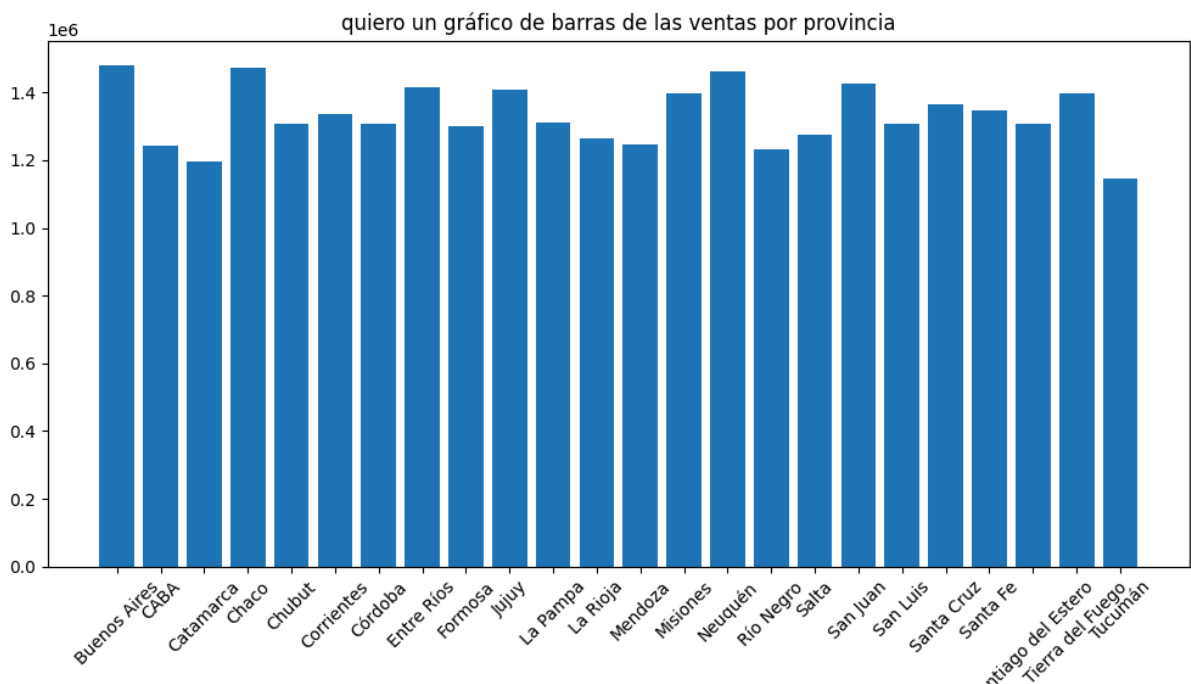
Las pruebas de memoria conversacional funcionaron perfectamente, donde la consulta 'licuadoras menos de 300 dólares' seguida por '¿Y cuáles tienen buen stock?' se enriqueció automáticamente a 'licuadoras con buen stock', demostrando la capacidad del sistema para mantener contexto conversacional de manera inteligente. La herramienta de analytics generó exitosamente gráficos dinámicos con consultas SQL como 'SELECT cliente\_provincia, SUM(cantidad) AS total\_ventas FROM ventas GROUP BY cliente\_provincia' junto con código matplotlib correspondiente para visualización automática.

El sistema híbrido demostró robustez excepcional: cuando se agotaron los límites de la API de Groq (5 requests por minuto en tier gratuito), el sistema automáticamente utilizó algoritmos de fallback inteligente manteniendo funcionalidad completa sin interrupciones perceptibles para el usuario. Esta arquitectura híbrida asegura disponibilidad continua del servicio independientemente de limitaciones externas.

## **Pruebas adicionales: Query SQL + gráfico**

Consulta: "quiero un gráfico de barras de las ventas por provincia"

SQL Ejecutado: SELECT cliente\_provincia, SUM(total) AS ventas\_totales FROM ventas GROUP BY cliente\_provincia



La consulta SQL fue construida completamente por el agente y se graficó en base a la consulta.

## Pipeline de Búsqueda Híbrida:

Para la base vectorial se implementó un pipeline de recuperación híbrida que combina múltiples estrategias de búsqueda:

1. Búsqueda Vectorial: Recupera fragmentos semánticamente similares a la consulta comparando embeddings en el espacio vectorial.
2. Búsqueda BM25: Recuperación basada en coincidencias léxicas, priorizando términos exactos y su frecuencia dentro de los documentos.
3. Fusión de Rankings (RRF): Combina ambos resultados mediante Reciprocal Rank Fusion para obtener un conjunto más robusto de candidatos.
4. Re-ranking con Cross-Encoder: Refina el orden de los resultados evaluando directamente cada par (consulta, fragmento) para priorizar los más relevantes.

## Mejoras propuestas:

Las principales mejoras futuras se relacionan con la incorporación de datos reales de fabricantes, lo que permitiría responder consultas técnicas con mayor precisión. También se propone extender la memoria conversacional para manejar múltiples restricciones simultáneas y agregar mecanismos automáticos de evaluación de calidad que permitan monitorear el rendimiento del sistema de forma continua.

## Conclusión:

El trabajo demuestra que es posible integrar modelos de lenguaje, técnicas clásicas de inteligencia artificial y múltiples estructuras de datos dentro de un pipeline RAG coherente y extensible. La evolución hacia un agente ReAct completamente dinámico evidencia el potencial de estas arquitecturas para construir sistemas flexibles, robustos y adaptables.

En conjunto, el proyecto muestra que, incluso con recursos moderados, se pueden desarrollar soluciones capaces de razonar, seleccionar herramientas, generar código y recuperar información de forma eficiente, ofreciendo una experiencia conversacional avanzada y contextualizada.