Instituto Tecnológico y de Estudios Superiores de Monterrey.

Escuela de Ingenierías

Maestría en Inteligencia Artificial Aplicada – Proyecto Integrador Grupo 10, Equipo 47.

Profesores:

Dra. Grettel Barceló Alonso
Dr. Luis Eduardo Falcón Morales
Mtra. Verónica Sandra Guzmán de Valle
Dr. Gerardo Jesús Camacho González
Dr. Eusebio Vargas Estrada

Modelos Alternos

Equipo #47

Erick Eduardo Betancourt Del Angel	A01795545
Lucero Guadalupe Contreras Hernández	A01794502
Erik Morales Hinojosa	A01795110

Fecha de entrega: 26 de Octubre del 2025

Repositorio de GitHub:

https://github.com/erikmoralestec/proyecto-integrador-grafos-de-conocimiento-llm

I. Situación actual

En el avance anterior se trabajó la proyección incremental con IPCA sobre embeddings de Complaints (reducción a 256-D con parámetros persistidos para garantizar consistencia entre indexación e inferencia) y su articulación dentro del flujo de recuperación semántica y grafo.

En este avance, el enfoque se desplazó a fortalecer la representatividad y trazabilidad de los datos mediante estratificación por componente, normalización de identificadores (p. ej., CAMPNO) y depuración tipada (años, marcas/modelos), así como a optimizar el pipeline de embeddings (uso de GPU y ajuste de batch_size) y la ingesta al almacén vectorial en entorno local. De este modo, el documento presenta (II) las optimizaciones aplicadas y (III) los obstáculos y restricciones observados, con los resúmenes cuantitativos correspondientes en las Tablas 1–3.

El entregable abarca la preparación de datos, la optimización del pipeline de embeddings y la evaluación de viabilidad operativa de las piezas principales (almacén vectorial y base grafo) como base para flujos posteriores.

En cuanto a orquestación e integración de modelos (trabajadas en el avance anterior), se empleó LangChain como capa de coordinación para encapsular retrievers específicos (Qdrant) y consultas a grafo (Neo4j), y se incorporó GraphCypherQAChain para traducir preguntas a Cypher y recuperar subgrafos pertinentes al contexto.

Paralelamente, se exploró la integración de LLMs open-weight (Ej. LLaMA 3.3-70B vía ChatGroq) como mecanismo de razonamiento y generación con anclaje en las fuentes (documentos recuperados y subgrafos). La configuración se centró en: (i) inyección de contexto controlada (metadatos, delimitadores y formatos de evidencia para trazabilidad), (ii) diseño de prompts orientados a preguntas técnicas y relaciones causales, (iii) gestión del presupuesto de tokens para evitar truncamiento y (iv) criterios de validación de coherencia y consistencia con el conocimiento estructural.

Con ello, el sistema progresó desde un RAG puramente vectorial hacia un RAG híbrido vector-grafo-LLM, donde el LLM actúa como capa de explicación y composición de evidencias, manteniendo proveniencia y verificabilidad de las respuestas. La profundización de estos aspectos se desarrolla en las secciones IV–V.

II. Optimización de embeddings

A continuación se describen los ajustes de datos y de cómputo orientados a mejorar rendimiento, estabilidad y trazabilidad del pipeline.

2.1 Filtrado y normalización de datos

Se acotó el alcance a V (**Vehicles**) y T (**Tires**) y se excluyeron C/E (Child seats y Equipment) por no ser parte de los objetivos. Tras el filtrado: Complaints pasó de 2,137,711 a **513,314** registros (-76%), Recalls de 14,340 a 13,653 (-4.8%) e Investigations de 153,551 a 149,506 (-2.6%). En Complaints, V se preservó de 2,066,945 → 512,972, T de 40,758 → 342, y se excluyeron 30,008 registros y 727 marcas exclusivas de C/E (1.4% del total).

Se depuraron outliers y valores anómalos (p. ej., años fuera de rango) y se normalizaron identificadores clave que dificultaron el enlace entre **Recall↔Investigation** (en particular el **número de campaña**) estandarizando tipo, longitud y formato (sin notación científica) y corrigiendo padding o ceros a la izquierda. El preprocesamiento incluyó coerción tipada de campos (años como enteros; marcas/modelos como categóricos), armonización de encodings y consolidación de variantes léxicas en COMPDESC/COMP_L1. Este tratamiento garantiza reproducibilidad y reduce ambigüedad de joins.

2.2 Downsampling estratificado (25%)

Se aplicó muestreo estratificado por componente (COMPDESC/COMP_L1) con semilla fija y cuotas proporcionales, garantizando mínimos por clase. La representatividad se validó cuantitativamente: el error promedio de proporciones por componente se mantuvo ≈0.12% y el error máximo ≈1.76%, con cobertura temporal estable y alto overlap en Top-50 de marcas. Podemos ver en el resumen de las métricas y resultados obtenidos que se muestran en la Tabla 1 que este enfoque reduce la carga de cómputo y almacenamiento sin degradar la calidad de recuperación observada.

Métrica	Valor observado	Interpretación
Error promedio de proporciones	≈0.12%	Excelente preservación de clases
Error máximo de proporciones	≈1.76%	Sin sesgos relevantes
Overlap Top-50 marcas	≈86%	Estructura de marcas consistente
Cobertura temporal (años)	67 años presentes	Sin sesgo temporal

Tabla 1. Calidad del downsampling (25%)

2.3 Parámetros de embedding y rendimiento

En línea con el avance anterior, se mantuvieron configuraciones que demostraron impacto directo en el throughput de embebido: uso de GPU, ajuste dinámico de batch_size condicionado por la VRAM disponible y precisión mixta (fp16/bfloat16 cuando fue viable),

además de data loaders con prefetching y control de longitud de tokenización. Estas decisiones, conservadas por su eficacia, permitieron reducir ~4× el tiempo total de generación de embeddings sin afectar la fidelidad semántica y sostener una ejecución estable en colecciones voluminosas. La síntesis de tiempos de embebido se resume en la Tabla 3.

III. Obstáculos encontrados

3.1 Capacidad y tiempos en entornos cloud

Durante cargas masivas a Qdrant (nuestro vector store administrado), se observaron latencias por solicitud y episodios de throttling asociados a cuotas compartidas, con tiempos estimados de 6–7 horas para los 512K vectores en lotes pequeños.

En la base grafo administrada, el límite práctico de relaciones (~400K) se alcanzó con rapidez, lo que vuelve riesgosa la persistencia de relaciones derivadas (similitudes/expansiones) y proyecta la superación de cuota al sumar Investigations. Estos efectos y su contraste con la operación local se sintetizan en la Tabla 3, mientras que la proximidad a los límites de grafo se detallan en la Tabla 2.

Métrica	**Actual**
Nodos	≈49,825 / 200,000 (25%)
Relaciones	≈325,214 / 400,000 (81%)
Riesgo	Alto por cercanía al límite

Tabla 2. Capacidad de Neo4j Aura vs proyección

Aspecto	**Cloud (saturado)**	**Local (Docker/Desktop)**
Subida 512K vectores	~6–7 horas	~30–35 minutos
Latencia por request	50–100 ms	1–2 ms
Batch size efectivo	≤ 500 (throttling)	≥ 2000 (sin throttling)
Control de recursos	Limitado	Total
Costo	Potencial	Sin costo adicional

Tabla 3. Comparativa Cloud vs Local (tiempos y operación)

3.2 Calidad de identificadores de vínculos Recall↔Investigation (obstáculo y oportunidad)

Al analizar la concordancia entre Investigations y Recalls se constató que no todas las investigaciones están asociadas a una campaña de retiro; esta cobertura incompleta constituye un obstáculo para consultas puramente relacionales, ya que limita la trazabilidad directa entre quejas, investigaciones y campañas.

No obstante, el vacío también abre una oportunidad para que agentes operen como capa inferencial que proponga relaciones faltantes bajo criterios verificables (p. ej., coincidencia semántica en narrativas, proximidad temporal, Make/Model/Component, ubicación, y, cuando exista, congruencia de identificadores).

En términos operativos, esto sugiere un flujo con puntuación de confianza, proveniencia explícita y, cuando el umbral lo amerite, validación humana en el ciclo. Así, el sistema reconoce la limitación actual como un espacio de ampliación controlada del grafo—donde las relaciones propuestas por agentes quedan auditables y reversibles, manteniendo la integridad del modelo de datos mientras se incrementa su cobertura efectiva.

IV. Integración con LangChain

La modularidad y extensibilidad de LangChain lo posicionaron como la plataforma central para la orquestación del sistema propuesto. LangChain permitió abstraer las complejidades de componentes heterogéneos (vector stores, motores de grafos, LLMs y APIs) en una arquitectura interoperable y escalable, permitiendo iteraciones rápidas y mejora progresiva en cada notebook.

4.1. Aplicación Inicial (Notebook 8 – LangChain + FastAPI)

En la primera etapa del desarrollo, LangChain fue utilizado para:

- Implementar retrievers personalizados orientados a los documentos específicos de la NHTSA (recalls, complaints, investigations), empleando QdrantRetriever sobre una base de vectores generada con embeddings E5.
- Construir un backend funcional con FastAPI, con endpoints /ask (simple retrieval) y /ask_lc (retrieval + LLM), permitiendo el consumo del sistema vía HTTP.
- Integrar consultas básicas a Neo4j para obtener entidades relacionadas con las campañas de retiro.

A pesar de sus funcionalidades, esta versión inicial no integraba traversal grafo-semántico, y Neo4j era utilizado de forma aislada mediante consultas manuales. El sistema operaba exclusivamente como un RAG basado en similitud semántica, sin representar explícitamente las relaciones estructurales que existen entre entidades como campañas, fallas o fabricantes.

4.2. Expansión con Grafo Semántico (Notebook 10 – GraphRAG-Optimizado)

La etapa culminante del proyecto incorporó el uso de la clase GraphCypherQAChain de LangChain, habilitando:

- La traducción automática de preguntas en lenguaje natural a consultas Cypher, generadas dinámicamente a partir del contexto de la pregunta.
- La recuperación de subgrafos relevantes, basados en relaciones como SIMILAR_TO, CAUSES, AFFECTS_MODEL, que enriquecen la comprensión semántica de los datos.

Esta capacidad permitió evolucionar desde una recuperación de tipo unidimensional (basada exclusivamente en embeddings y distancias vectoriales) hacia un sistema **estructuralmente sensible**, que representa explícitamente relaciones entre entidades y que posibilita la ejecución de inferencias complejas mediante traversal del grafo. Como resultado, se logró una fusión semántica entre contexto vectorial y conocimiento relacional.

V. Integración de Modelos de Lenguaje (LLMs)

5.1. Selección del Modelo LLM

Para mantener el sistema accesible y replicable, se optó por un modelo de lenguaje open-weight, seleccionando LLaMA 3.3–70B vía la plataforma ChatGroq. Las razones para esta elección se resumen en la Tabla 1:

Criterio	**Justificación**
Precisión semántica	Capacidad de entender prompts técnicos y relaciones causales
Rendimiento	Bajos tiempos de inferencia en la nube
Multimodalidad contextual	Manejo simultáneo de texto enriquecido y conocimiento estructurado
Acceso libre	Independencia de proveedores propietarios

Tabla 4. Criterios de selección del modelo LLM

5.2. Integración Funcional

La integración funcional se implementó de manera progresiva desde el notebook 9 hasta el 10, siguiendo un pipeline estructurado:

1. **Recuperación de contexto semántico** desde Qdrant (hasta 8 documentos por pregunta), utilizando retrievers optimizados con filtros por marca, modelo y año.
2. **Formateo enriquecido del contexto**, utilizando metadatos clave y delimitadores (SOURCE:, CAMPAIGN:, VEHICLE:) que orientan al LLM sobre cómo interpretar la entrada.
3. **Inyección de contexto y generación de prompt**, considerando longitud del token input para evitar truncamiento de información crítica.
4. **Generación de respuesta**, validación de coherencia semántica, y refinamiento mediante análisis de relaciones causales y consistencia de salida.

En el notebook 10, se extendió el pipeline para combinar simultáneamente:

- Documentos recuperados por similitud vectorial (Qdrant).
- Subgrafos obtenidos vía traversal semántico (Neo4j).
- Razonamiento generado por LLM con trazabilidad hacia los elementos de origen.

5.3. Beneficios Observados

La incorporación de LLMs incrementó sustancialmente la capacidad del sistema para:

- Explicar decisiones de manera natural y estructurada.
- Relacionar múltiples documentos dispares y conectarlos en una narrativa lógica.
- Inferir causalidades, como el impacto de fallas en campañas futuras o vínculos entre incidentes similares.
- Responder preguntas abiertas complejas que combinan múltiples dimensiones (temporalidad, fabricante, tipo de falla, afectación geográfica).

Capacidad del Sistema	**Sin LLM (Notebook 8)**	**Con LLM + Grafo (Notebook 10)**
Recuperación semántica	✅ Sí	✅ Sí
Comprensión contextual compleja	❌ No	✅ Sí
Relación entre múltiples documentos	❌ Parcial	✅ Sí
Justificación estructurada	❌ No	✅ Sí
Explicación de causalidades	❌ No	✅ Sí

Tabla 5. Comparativa de capacidades antes y después de integrar LLMs

VI. Próximos pasos

La evolución del grafo se orientará a un esquema mínimo materializado que preserve únicamente las relaciones canónicas basadas en identificadores normalizados (por ejemplo, vínculos entre quejas y campañas cuando exista correspondencia inequívoca por CAMPNO), trasladando toda relación inferida, como las similitudes, al plano de consulta mediante virtualización. Sobre este diseño, proponemos una capa de agentes que opere como “relacionador” entre Investigations y Recalls allí donde la cobertura actual es incompleta: el agente generará candidatos de enlace a partir de una combinación de similitud semántica en la narrativa de los documentos, proximidad temporal de eventos, coincidencias en Make/Model/Component, y, cuando exista, congruencia de identificadores y georreferencias.

Cada propuesta incluirá un puntaje de confianza, trazabilidad de evidencias (fragmentos textuales y subgrafos consultados) y un registro de proveniencia que permita auditoría posterior; de esta forma, las nuevas aristas se introducen de manera auditable y reversible, respetando la integridad del modelo de datos.