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: 19 de Octubre del 2025

Repositorio de GitHub:

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

I. Encaje de datasets

Esta sección documenta el proceso de alineación semántica y estructural entre los diferentes conjuntos de datos utilizados: *recalls* (llamados a revisión), *complaints* (quejas) e *investigations* (investigaciones). El objetivo del encaje fue identificar entidades equivalentes o relacionadas entre estos dominios heterogéneos, con el fin de lograr una integración coherente y robusta dentro del grafo de conocimiento.

1.1 Tipos de Encajes Realizados

Los encajes se implementaron a través de una combinación de reglas estructuradas y técnicas de representación semántica:

- **Encajes estructurales por identificador**: Campos clave como MAKE, MODEL, YEAR, COMPONENT, CAMP_NO, ACTION_NUMBER y VEHICLE_KEY fueron estandarizados y mapeados a identificadores únicos (*_ID), lo que permitió establecer relaciones deterministas entre entidades provenientes de distintos archivos.
- **Encajes semánticos por similitud textual**: Se aplicaron procesos de normalización tales como slugify, conversión a minúsculas, eliminación de caracteres especiales y reducción de stopwords para homogeneizar cadenas de texto, especialmente en nombres de fabricantes, modelos y componentes.
- **Encajes vectoriales mediante embeddings**: Utilizando el modelo e5-multilingual-large, se transformaron descripciones textuales narrativas (como defectos reportados y acciones correctivas) en vectores de alta dimensionalidad. A partir de estos vectores, se calculó la similitud semántica entre pares de entidades, lo que permitió detectar correspondencias incluso cuando el texto no era idéntico.
- **Encajes jerárquicos**: El campo COMPNAME fue descompuesto en tres niveles jerárquicos (COMP_L1, COMP_L2, COMP_L3), representando la arquitectura de componentes del vehículo. Esto habilitó una representación granular y semánticamente rica dentro del modelo de grafo.

1.2 Herramientas y Notebooks Asociados

Los procesos de encaje fueron implementados en notebooks independientes, cada uno enfocado en una categoría de datos:

- **3.1-Encajes de recalls.ipynb**: Encaje de campañas de retiro (CAMP_NO) con fabricantes, modelos, componentes y años modelo.
- **3.2-Encajes de complaints.ipynb**: Encaje de quejas (COMPL_ID) con identificadores vehiculares y características del modelo.
- **3.3 Encajes de investigations**: Encaje entre investigaciones (ACTION_NUMBER) y otras entidades vinculadas como *recalls* y *complaints*.

Todos los procesos incluyeron mecanismos de validación como la detección de duplicados, verificación de tipos de dato, control de valores nulos y métricas auxiliares (por ejemplo, longitud de texto).

1.3 Embeddings en complaints y reducción de dimensión

Para habilitar búsqueda semántica sobre narrativas de quejas, cada complaint se transforma en un vector denso que captura significado (embedding). Utilizamos un encoder moderno de frases (estilo bi-encoder) para mapear textos a un espacio vectorial donde proximidad ≈ similitud semántica. Este enfoque, inspirado en Sentence-BERT, permite recuperar complaints semánticamente afines a una consulta, aún si las palabras no coinciden exactamente.

El embedding "nativo" tiene 1024 dimensiones. Sin embargo, indexar millones de vectores de 1024-D encarece memoria y latencia. Por eso aplicamos reducción con Incremental PCA (IPCA) sobre complaints únicamente, bajando a 256-D. IPCA es una variante de PCA que procesa en bloques (streaming), por lo que es adecuada cuando el dataset no cabe en RAM y hay que entrenar por lotes.

Entrenamos IPCA una sola vez y guardamos sus parámetros (componentes y media); así, cualquier consulta futura se proyecta con el mismo IPCA antes de buscar en la colección de complaints. Esta consistencia entre índice y consulta evita errores de "dimensión esperada" y, más importante, mantiene la geometría global suficiente para conservar vecinos semánticos útiles, pero a una fracción del costo.

Elegimos 256-D por equilibrio empírico entre precisión y costo. El resto de colecciones (recalls e investigations) se mantienen en 1024-D; de este modo no pierden señal cuando el volumen es menor. En tiempo de consulta, la ruta es: (1) generar embedding 1024-D; (2) si buscamos en complaints, proyectar con IPCA a 256-D; (3) consultar en el índice vectorial.

1.3.1 Pipeline para complaints: IPCA → LSH → MiniBatchKMeans

El objetivo es reducir volumen y coste de cómputo sin perder la señal semántica que hace útiles a los embeddings. El flujo trabaja en tres etapas:

1. IPCA reduce las dimensiones de los embeddings (1024→256) de forma incremental (por lotes), compatible con conjuntos muy grandes.
2. LSH (estilo SimHash) genera firmas binarias y agrupa por "cubetas" a puntos muy parecidos; dentro de cada cubeta elegimos un representante. Esta etapa actúa como downsampling semántico local y deduplicación aproximada.
3. MiniBatchKMeans (dos pasadas) sobre los vectores ya reducidos obtiene representantes globales (centroides cercanos) a escala, con RAM acotada y entrenamiento rápido.

El resultado es un subconjunto representativo de complaints que preserva la diversidad temática y reduce la redundancia, listo para indexación en la base vectorial o para construir grafos manejables.

A) Reducción incremental con IPCA

Qué resuelve. Mantener la estructura global de los datos con menos dimensiones, reduciendo memoria, latencia y almacenamiento. IPCA es PCA “en streaming”: puedes entrenar con minibatches sucesivos y sin tener toda la matriz en RAM.

Cómo se aplica.

- Se entrena IPCA con lotes (p. ej., 100k filas) sobre embeddings de 1024-D.
- Se persisten parámetros (components_, mean_, varianzas, etc.) en un archivo único (p. ej., ipca_model.npz).
- Se transforma cada shard de complaints a 256-D, escribiendo .npy con cabecera válida y su meta sincronizada (id, make, model, year, etc.).
- En consulta, cualquier vector 1024-D se proyecta con el mismo IPCA antes de buscar en la colección que espera 256-D.

Por qué funciona.
En textos embebidos, gran parte de la variación útil se concentra en pocas direcciones. IPCA conserva las componentes principales que explican la mayor varianza; en práctica, 256-D suele retener suficiente estructura para ANN search y análisis posteriores.

B) LSH (SimHash-like) para deduplicación y downsampling local

Qué resuelve.
Escenarios con mucha redundancia (múltiples quejas casi idénticas) o temas masivos. Hacer k-means a todo el universo puede ser caro; LSH ofrece un pre-agrupamiento aproximado y muy barato.

Cómo se aplica.

1. Se genera al azar una matriz de proyección $d \times b$ y se computa el signo del producto $X \cdot P$ para cada punto reducido: una firma binaria de b bits (SimHash).
2. Se “parte” la firma en bandas (Bands × Rows) y se indexa en cubetas por banda. Puntos que comparten la misma subfirma caen en la misma cubeta y son candidatos muy similares bajo coseno/Hamming.
3. Por cada cubeta se escoge un representante (p. ej., el primero o el más frecuente), marcando el resto como cubiertos.
4. Se escribe un CSV de representantes por grupo (5 shards por lote, etc.), y luego se consolida en un archivo global.

Qué aporta.
Al quedarte con 1 representante por cubeta, haces un downsampling semántico que reduce el tamaño sin destruir la diversidad. Además, esta etapa es lineal y completamente paralelizable por shards.

C) MiniBatchKMeans (2-pasos) para representantes globales

Qué resuelve. Obtener representantes a nivel global (no sólo por cubeta LSH), con cientos de miles o millones de puntos y poca RAM.

Plan 2-pasos.

1. Fit incremental: con partial_fit sobre shards reducidos (256-D), se entrena MiniBatchKMeans con K grande (p. ej., 20 000) y batch_size moderado (p. ej., 4096).
2. Assign + selección: se etiqueta cada punto con su centro y se queda el más cercano por clúster (mínima distancia al centro). Así se construye una tabla final de representantes globales con sus metadatos (ids, make/model/year, shard y fila).

Ventajas.

- Tiempo: mini-batches evitan pasar toda la matriz por RAM y aceleran entrenamiento.
- Estabilidad: con k-means++ en MiniBatch, la inicialización es robusta.
- Interpretabilidad: puedes inspeccionar 1 punto real por clúster (no sólo el centroide).

Encaje entre etapas y consistencia

1. Orden: Embeddings (1024) → IPCA (256) → LSH (firma/bandas) → MiniBatchKMeans (256).
2. Persistencia: el mismo IPCA debe usarse para transformación e inferencia; cambia IPCA ⇒ reindexa.
3. Metadatos: cada .npy reducido debe acompañarse de su meta.parquet para trazabilidad (id, _h, shard, row_in_shard).
4. Agnóstico a consultas: el LSH/MBKMeans se ejecuta offline para construir subconjuntos; en consulta, sólo se proyecta (IPCA) y se consulta la colección adecuada (256 vs 1024).

Elección de hiperparámetros (criterios prácticos)

- Dimensión IPCA (p. ej., 256): balancea recall semántico vs tamaño/latencia.
- LSH: bands × rows fija el umbral de colisión. Más filas por banda ⇒ candidatos más estrictos; más bandas ⇒ mayor recall.
- K en MiniBatchKMeans: apunta a #de representantes finales que quieres (p. ej., 20 000); ajusta batch_size a la RAM.
- Validación: muestrea pares “quedó/descartado” y revisa diversidad de componentes y distribuciones por make/model/year para evitar sesgos.

Fallos comunes y cómo evitarlos

- Dimensión inconsistente entre índice y consulta (256 vs 1024): proyecta todas las consultas con el IPCA correcto antes de buscar en complaints.
- Archivos reducidos sin cabecera .npy (memmap crudo): reescribir con open_memmap para cabecera válida.
- Desfase npy–meta: comprobar que el número de filas coincida; recortar si difiere.
- Clústeres vacíos en MiniBatch: ajustar reassignment_ratio y repetir una pasada de partial_fit

1.3.2 Por qué este pipeline es apropiado para complaints

Los textos de quejas tienden a repetirse (mismos fallos narrados de forma parecida), lo que provoca alta redundancia en el espacio vectorial. IPCA reduce coste sin destruir la proximidad semántica; LSH elimina duplicación local casi gratis; MiniBatchKMeans sintetiza el conjunto en representantes interpretables que preservan la variedad temática. Así, el grafo que construyes y la base vectorial que consultas son manejables y expresivos a la vez.

II. Integración Qdrant ↔ Neo4j (búsqueda semántica + grafo canónico)

Separamos responsabilidades: Qdrant resuelve "¿qué elementos son semánticamente más cercanos a mi consulta?" y Neo4j responde "¿cómo se relacionan esos elementos entre sí y con el resto del dominio?". El orquestador ask() hace lo siguiente: genera el embedding de la consulta, interroga la colección pertinente en Qdrant y recupera payloads (IDs y metadatos mínimos). Con esa lista de IDs, ask() busca en Neo4j los nodos completos y sus relaciones explícitas:

OF_MAKE y OF_MODEL (normalizan marca y modelo),
MENTIONS (componentes mencionados),
RELATES_TO (vínculo regulatorio Investigation → Recall cuando existe).

Como parte del análisis, añadimos un enlace inferido y auditable entre complaints y recalls llamado MATCH_CR: une quejas y retiros cuando comparten make y model y el año difiere a lo más ±1 (o ±2 en modo más laxo). No sustituye a RELATES_TO; sirve para descubrir candidatos coherentes con la cronología del vehículo. De manera análoga, calculamos INV_CAND (investigación ↔ recall candidato por "familia" de atributos: make, model o component). Este enlace se dibuja gris para dejar claro que es sugerido, no declarado.

Esta arquitectura tiene ventajas claras: los top-K semánticos en Qdrant limitan el problema global, y Neo4j reconstituye el subgrafo con su semántica de dominio (marcas, modelos, componentes, vínculos regulatorios). La combinación permite empezar con una sola frase ("airbag sensor failure") y terminar con un subgrafo navegable que incluye quejas, retiros, investigaciones y entidades maestras.

III. Diseno del Grafo

El grafo de conocimiento fue diseñado para representar las relaciones existentes entre los distintos actores del ecosistema automotriz regulado. El modelo captura interacciones regulatorias, técnicas y semánticas, utilizando una ontología que permite análisis de causa-efecto, agrupamiento semántico, trazabilidad y recuperación de información.

3.1 Tipos de Nodos

El grafo está compuesto por los siguientes tipos de nodos:

- Recall: campaña de retiro por seguridad; contiene identificador regulatorio, fecha, resumen/subject y metadatos como make, model y (cuando existe) year.
- Complaint: queja formal de consumidores; incluye make, model, year y texto libre (base de embeddings).
- Investigation: investigación de la autoridad; almacena identificador, fechas, subject/summary y, cuando existe, relación directa al recall.
- Make: fabricante normalizado.
- Model: modelo comercial normalizado.
- Component: sistema/parte afectada (p. ej., AIR BAGS, SERVICE BRAKES).

Decisión de modelado: ModelYear se maneja como propiedad (year) en Complaint y Recall para reducir cardinalidad y simplificar las consultas temporales; cuando se requiere razonamiento explícito por año, se puede proyectar un nodo ModelYear mediante view o materialización parcial.

Cada nodo posee un identificador único y atributos “listas para grafo” (strings normalizados en mayúsculas, fechas parseadas, y claves textuales limpias para tooltips y filtros).

3.2 Tipos de Relaciones

Las aristas codifican vínculos estructurales del dominio y evidencias derivadas:

Estructurales normalizadas

- (Complaint|Recall|Investigation)-[:OF_MAKE]->(Make)
- (Complaint|Recall|Investigation)-[:OF_MODEL]->(Model)
- (Complaint|Recall|Investigation)-[:MENTIONS]->(Component)

Regulatorias explícitas

- (Investigation)-[:RELATES_TO]->(Recall) cuando la fuente lo declara.

Inferencias auditables (basadas en datos estructurados)

- (Complaint)-[:MATCH_CR {delta_year}]->(Recall) si coinciden make y model y el año difiere a lo más ±1 (o ±2 en modo laxo). Esta arista no reemplaza un vínculo regulatorio; sirve para hipótesis y priorización.
- (Investigation)-[:INV_CAND]->(Recall) si comparten familia (make, model o component). Es un candidato que sugiere proximidad temática cuando no existe RELATES_TO.

Las relaciones puramente vectoriales tipo (:Complaint)-[:SIMILAR_TO]->(:Complaint) quedan opcionales y fuera del “núcleo regulatorio”. En su lugar, las usamos como mecanismo de descubrimiento (top-K por Qdrant) y luego se traducen en aristas estructurales (MATCH_CR, INV_CAND) para mantener el grafo explicable.

IV. Construcción del Grafo

La construcción combina exportación/ingestión de entidades y orquestación semántica con Qdrant. Neo4j es el “estado canónico” del grafo; Qdrant es el “catálogo vectorial” para recuperar candidatos semánticos.

4.1 Exportación a CSV

Los nodos y relaciones se exportan por tipos (p. ej., nodes_Make.csv, nodes_Model.csv, nodes_Component.csv, nodes_Complaint.csv) y relaciones (rel_OF_MAKE.csv, rel_OF_MODEL.csv, rel_MENTIONS.csv, rel_RELATES_TO.csv). Durante la exportación se:

- Normalizan claves (make/model/component),
- Alinean identificadores,
- Validan tipos (fechas, enteros),
- Separan texto libre (para embeddings) de claves normalizadas (para grafo).

4.2 Ingestión en Neo4j

La ingestión usa LOAD CSV WITH HEADERS + MERGE (idempotente) y restricciones/índices para integridad y rendimiento:

```
# Nodos

LOAD CSV WITH HEADERS FROM 'file:///nodes_Model.csv' AS row
MERGE (:Model {MODEL_ID: row.MODEL_ID})
  ON CREATE SET .name = row.MODEL
LOAD CSV WITH HEADERS FROM 'file:///nodes_Make.csv' AS row
MERGE (:Make {MAKE_ID: row.MAKE_ID})
  ON CREATE SET .name = row.MAKE

# Relaciones

LOAD CSV WITH HEADERS FROM 'file:///rel_OF_MODEL.csv' AS row
MATCH (c:Complaint {QID: row.QID})
MATCH (m:Model {MODEL_ID: row.MODEL_ID})
MERGE (c)-[:OF_MODEL]->(m);
```

Se definen índices/uniqueness en identificadores regulatorios (Recall.id, Investigation.id) y claves maestras (Make.name, Model.name, Component.name).

4.3 Relaciones Vectoriales

El cálculo de relaciones semánticas tipo SIMILAR_TO entre entidades textuales se realizó mediante técnicas de similitud de embeddings. Se emplearon:

- El modelo e5-multilingual-large para generar vectores semánticos.
- Algoritmos como FAISS (con soporte para CPU/GPU) y NearestNeighbors para encontrar vecinos más cercanos.
- Un umbral de similitud (score > 0.75) para establecer relaciones sólidas entre descripciones conceptualmente similares.

Este enfoque permite consultas basadas en significado, agrupamiento de quejas y exploración de patrones emergentes.

4.4 Señal semántica y pipeline de reducción (Complaints)

Para operar a escala, el texto de complaints se embebe en 1024-D y se pasa por el pipeline IPCA → LSH → MiniBatchKMeans:

1. IPCA (Incremental PCA): reduce 1024→256 de forma incremental y reproducible (guardando components_ y mean_).
2. LSH tipo SimHash: genera firmas binarias y downsampling local (1 representante por cubeta) para disminuir redundancia.
3. MiniBatchKMeans 2-pasos: partial_fit sobre shards → asignación y selección del más cercano al centroide por clúster (representantes globales).

El resultado son complaints representativos con dimensión 256-D listos para indexar y consultar eficientemente.

V. Visualización y lectura

La visualización muestra, en un mismo plano, la evidencia semántica recuperada por Qdrant y las relaciones canónicas almacenadas en Neo4j. La lógica es: Qdrant propone candidatos por significado (top-K) y Neo4j otorga estructura, trazabilidad y reglas explícitas (marcas, modelos, componentes y vínculos regulatorios). Así, una sola consulta textual (p. ej., “airbag sensor failure”) se convierte en un subgrafo navegable que conecta complaints, recalls, investigations y entidades maestras.

5.1. Nodos (qué ves y cómo leerlos)

En la vista final, cada tipo de entidad aparece con forma y color consistentes:

- Complaint — punto gris. Quejas recuperadas por similitud semántica (Qdrant) y confirmadas en Neo4j (ID, make, model, year, componente).
- Recall — caja morada. Campañas de retiro con su identificador regulatorio, fechas y resumen/subject.
- Investigation — rombo ámbar. Investigaciones de la autoridad; cuando existe, pueden declarar una relación regulatoria hacia un recall.
- Make — caja verde. Fabricante normalizado (clave maestra).
- Model — caja cian. Modelo normalizado (clave maestra).
- Component — punto rosa. Sistema o parte afectada (p. ej., AIR BAGS, SERVICE BRAKES).

Una buena lectura inicial es buscar hubs: marcas o modelos con muchos incidentes conectados suelen concentrar evidencia. Los agrupamientos alrededor de un componente (puntos rosas) señalan focos temáticos claros.

5.2. Aristas (qué significa cada color)

Las aristas codifican relaciones explícitas del dominio y evidencias inferidas y auditables:

- OF_MAKE (verde): une Complaint/Recall/Investigation con su Make (normalización de fabricante).
- OF_MODEL (cian): une Complaint/Recall/Investigation con su Model (normalización de modelo).
- MENTIONS (rosa): une Complaint/Recall/Investigation con Component (tema técnico mencionado).
- RELATES_TO (azul): vínculo regulatorio explícito Investigation → Recall (declarado en la fuente).
- MATCH_CR (tráfico): Complaint ↔ Recall inferido cuando coinciden make y model y la diferencia de año es pequeña.
    - Verde: Δaño = 0
    - Naranja: Δaño = ±1

    - Rojo: Δaño = ±2 (modo laxo)
      Esta arista propone hipótesis plausibles (no sustituye RELATES_TO) y ayuda a priorizar revisión.
- INV_CAND (gris claro): candidato Investigation ↔ Recall si comparten familia de atributos (misma marca o modelo o componente). Sirve para reducir huérfanos y sugerir rutas de inspección cuando no hay RELATES_TO declarado.

En conjunto, los colores permiten distinguir de un vistazo qué es canónico (verde/cian/rosa/azul) y qué es sugerido (tráfico para complaint–recall por cronología, gris para investigation–recall por familia).

5.3. ¿Por qué aparecen “huérfanos” y cómo mitigarlos?

Es normal observar nodos aislados en un subgrafo muestreado. Las causas típicas son:

- Falta de relación explícita: por ejemplo, una investigation sin RELATES_TO registrado aún; o un recall que no llegó a cruzarse con complaints dentro del corte. (Existen productos de carros que no precisamente son vehículos, como por ejemplo: baterías)
- Datos incompletos: si una entidad carece de make/model/component, no se pueden dibujar OF_* ni MENTIONS, y pierde “puentes” hacia el resto.
- Corte de muestreo (top-K): si el recall relevante no entró en los top-K de Qdrant, la investigation puede quedar sola aunque sea pertinente.

Dos decisiones del pipeline ya ayudan a reducir huérfanos sin “inventar” relaciones:

- Asegurar nodos referenciados: si una investigation declara RELATES_TO un recall, ese recall se incluye aunque no haya salido en el top-K textual; así no se corta la trazabilidad.
- Candidatos por familia (INV_CAND): cuando no hay RELATES_TO, conectar Investigation ↔ Recall por make/model/component ofrece sugerencias auditables que suelen cerrar islas sin forzar vínculos normativos.

Con estos dos mecanismos, la visual se vuelve explicable y accionable: lo explícito (azul/verde/cian/rosa) cuenta la historia regulatoria; lo inferido (tráfico/gris) propone dónde mirar después. Si se necesita más cobertura, basta aumentar K en Qdrant o expandir desde un nodo por RELATES_TO en Neo4j para completar vecindarios relevantes.

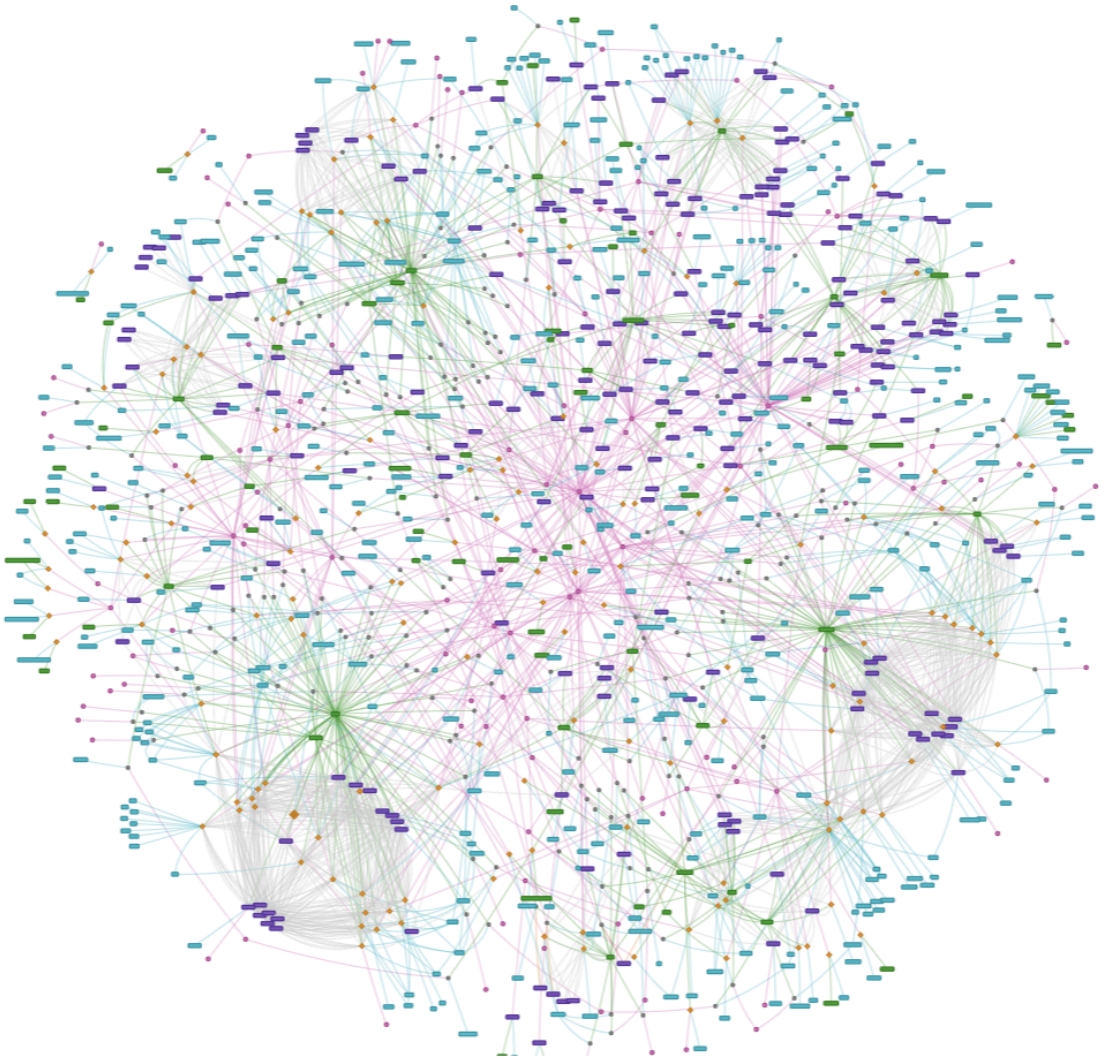

Figura 5.1. Evidencia integrada para “airbag sensor failure”.
Visualización interactiva disponible en: Grafo interactivo (https://raw.githubusercontent.com/erikmoralestec/proyecto-integrador-grafos-de-conocimiento-llm/main/graph_unificado_arc.html). Nodos: Complaint (gris), Recall (morado), Investigation (ámbar), Make (verde), Model (cian), Component (rosa). Aristas: OF_MAKE (verde), OF_MODEL (cian), MENTIONS (rosa), RELATES_TO (azul), MATCH_CR (tráfico según Δaño), INV_CAND (gris claro).

V. Conclusiones y próximos pasos

El presente trabajo ha logrado consolidar una arquitectura de grafo de conocimiento que integra de manera estructurada y semántica tres fuentes de información críticas del dominio automotriz: *recalls*, *complaints* e *investigations*. A través de un enfoque sistemático de exploración, limpieza, encaje y exportación, se ha construido un modelo relacional capaz de representar no solo los vínculos explícitos entre entidades (por ejemplo, campañas y modelos afectados), sino también relaciones latentes inferidas mediante técnicas de procesamiento de lenguaje natural.

La utilización de embeddings del modelo e5-multilingual-large permitió enriquecer el grafo con aristas semánticas entre descripciones textuales, abriendo la puerta a análisis avanzados de similitud, detección de patrones y agrupamiento de casos.

La modularidad del pipeline, basada en notebooks independientes para cada etapa, permite mantener flexibilidad y escalabilidad, facilitando la incorporación de nuevas fuentes o ajustes en la ontología del grafo.

Como etapa siguiente, se propone la incorporación de agentes inteligentes basados en *frameworks como LangChain* o *OpenAI Function Agents*, que puedan interactuar con el grafo de conocimiento para:

- Realizar consultas guiadas y explicativas sobre relaciones complejas o poco evidentes.
- Evaluar la coherencia y validez semántica de relaciones generadas automáticamente, especialmente aquellas inferidas por similitud de embeddings.
- Proporcionar un entorno conversacional o interfaz API que facilite la exploración del grafo por parte de usuarios técnicos y no técnicos.

Este enfoque orientado a agentes permitirá cerrar el ciclo completo de análisis: desde la integración de datos heterogéneos hasta su interpretación automatizada mediante razonamiento contextual y consulta inteligente.

VI. Referencias:

NHTSA datasets (Complaints, Investigations, Recalls).

Documentación técnica (Hugging Face, Multilingual E5 Text Embeddings, SentenceTransformers).

Charikar, M. (2002). Similarity estimation techniques from rounding algorithms. In Proceedings of the 34th Annual ACM Symposium on Theory of Computing (pp. 380–388). ACM. https://doi.org/10.1145/509907.509965

Gionis, A., Indyk, P., & Motwani, P. (1999). Similarity search in high dimensions via hashing. In Proceedings of the 25th VLDB (pp. 518–529). Morgan Kaufmann.

Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis: A review and recent developments. Philosophical Transactions of the Royal Society A, 374(2065), 20150202. https://doi.org/10.1098/rsta.2015.0202

Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using HNSW. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(4), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Proceedings of EMNLP-IJCNLP (pp. 3982–3992). ACL. https://doi.org/10.48550/arXiv.1908.10084

Sculley, D. (2010). Web-scale k-means clustering. In Proceedings of WWW ’10 (pp. 1177–1178). ACM. https://doi.org/10.1145/1772690.1772862