Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Trabajo Práctico - NLP

Rosito, Valentín R-4662/1

Estructura del Proyecto

A continuación, se detalla la organización del repositorio y la función de los archivos y directorios principales utilizados en el desarrollo del proyecto:

1.rag_flow.ipynb: Notebook que contiene la solución al "Problema 1". Implementa un sistema RAG con un flujo de ejecución simple, integrando métodos de búsqueda vectorial, tabular y de grafos, coordinados por un mecanismo de enrutamiento básico para responder consultas.

2.rag_agent.ipynb: Notebook correspondiente al "Problema 2". Despliega un flujo agéntico avanzado (RAG Agent) que utiliza un Agente ReAct. Este agente tiene la capacidad de razonar y orquestar consultas complejas combinando inteligentemente las tres bases de datos (vectorial, tabular y grafos).

data/: Directorio principal de almacenamiento de información. Se encuentra subdividido en **raw/**, que contiene los datos originales de los electrodomésticos sin procesar, y **processed/**, donde se almacenan los datos limpios y los embeddings generados listos para el proceso de recuperación.

pyproject.toml: Archivo de configuración técnica que define las dependencias del proyecto y las especificaciones del entorno, diseñado para ser gestionado mediante la herramienta `uv`.

.env: Archivo destinado a las variables de entorno locales. Su función principal es alojar de manera segura la `GOOGLE_API_KEY`, necesaria para autenticar y consumir los servicios de Google Generative AI.

Guía de Ejecución y Despliegue

Esta sección describe los pasos técnicos necesarios para configurar el entorno y ejecutar el proyecto localmente.

Requisitos Previos

- **Python:** Versión ≥ 3.10 y ≤ 3.13.
- **Gestor de paquetes:** uv.
- **Motor de Grafos:** Docker (para ejecutar el contenedor de Memgraph).
- **Credenciales:** API Key de Google AI Studio.

Procedimiento de Instalación

1. Instalación del gestor uv Si la herramienta no se encuentra en el sistema, ejecutar:

- # macOS/Linux
- curl -LsSf https://astral.sh/uv/install.sh | sh
-
- # O alternativamente con pip
- pip install uv

2. Clonado del repositorio Descargar el código fuente y acceder al directorio:

- cd /ruta/del/proyecto

3. Configuración del Entorno Virtual Crear y activar el entorno virtual para aislar las dependencias:

- # Crear entorno
- uv venv
-
- # Activar entorno (macOS/Linux)
- source .venv/bin/activate
-
- # Activar entorno (Windows)
- .venv\Scripts\activate

4. Instalación de Dependencias Instalar las librerías definidas en `pyproject.toml`:

- uv pip install -e .

5. Despliegue de Base de Datos de Grafos (Memgraph) Para habilitar la funcionalidad de grafos, es necesario ejecutar Memgraph mediante Docker. Asegúrese de que Docker Desktop esté corriendo y ejecute:

- docker run -p 7687:7687 memgraph/memgraph

  Comandos de gestión para Docker:

  - Ver logs: `docker logs memgraph`
  - Detener servicio: `docker stop memgraph`
  - Reiniciar servicio: `docker start memgraph`

6. Configuración de Variables de Entorno

Crear un archivo `.env` en la raíz del proyecto e ingresar la clave de API:

- GOOGLE_API_KEY=tu_api_key_aqui

Nota: Se recomienda utilizar una cuenta nueva en Google AI Studio para aprovechar el crédito gratuito (aprox. $300 USD por 90 días), lo cual incrementa el límite de peticiones por minuto (RPM) y reduce la latencia.

Arquitectura y Resultados del Sistema RAG

1. Arquitectura de Recuperación (RAG Flow)

Esta sección detalla las decisiones de diseño e implementación correspondientes al **Notebook 1**, enfocado en el pipeline de recuperación de información.

1.1. Decisiones de Infraestructura y Base de Datos

Almacenamiento Vectorial: ChromaDB

Se seleccionó **ChromaDB** debido a su practicidad, capacidad nativa para gestionar embeddings y su modelo de persistencia local, lo que elimina dependencias de servicios en la nube para esta capa.

- **Modelo de Embedding:** Se utilizó el modelo por defecto `all-MiniLM-L6-v2` (384 dimensiones). Este modelo ofrece una capacidad suficiente para la documentación técnica procesada y permitió agilizar el desarrollo sin requerir claves de API adicionales ni infraestructura de GPU dedicada.

Base de Datos de Grafos: Memgraph con Cypher

La implementación de una base de datos de grafos fue **fundamental** para modelar correctamente las relaciones complejas entre productos (e.g., `SIMILAR_TO`, `RECOMENDADO_CON`, `ALTERNATIVA_DE`).

- **Selección de Memgraph:** Se eligió por su compatibilidad del 100% con los drivers de Neo4j, su facilidad de despliegue mediante Docker y su rendimiento optimizado al operar en memoria.
- **Importancia del Lenguaje Cypher:** El soporte de Cypher es esencial en un entorno de LLMs por cuatro razones críticas:

1. **Generación Dinámica:** El LLM traduce lenguaje natural a Cypher de manera robusta.
2. **Sintaxis Declarativa:** Facilita la generación y validación frente a APIs imperativas.
3. **Estándar de Industria:** Al ser el estándar de facto (Neo4j), asegura mantenibilidad.
4. **Expresividad:** Permite consultas complejas (múltiples saltos, agregaciones) en pocas líneas de código.

1.2. Clasificador de Intención

Para el enrutamiento de consultas, se realizó una evaluación comparativa entre un modelo clásico (SVM + TF-IDF) y un enfoque basado en LLM Few-Shot.

- **Resultados:** Aunque el SVM demostró mayor velocidad, presentó una confusión crítica entre las categorías tabular y de grafo (F1-Score del 50% en tabular).
- **Decisión:** Se implementó el **LLM Few-Shot** (88.9% de accuracy global), logrando una mejora de **18.5 puntos porcentuales** sobre el SVM. Esta consistencia (88.9% F1-Score en todas las categorías) justifica el costo operativo marginal ($0.10/1M tokens).

1.3. Modelo de Lenguaje: Google Gemini 2.5 Flash

Gemini 2.5 Flash fue seleccionado como el motor de inferencia óptimo, destacando por ser la opción de mayor calidad en el segmento de costo reducido:

- **Alta Precisión:** Genera consultas SQL/Cypher válidas en más del 95% de los casos.
- **Eficiencia de Costos:** $0.10 por 1M de tokens, con un tier gratuito diario.
- **Contexto y Latencia:** Ventana de 1M de tokens para documentación extensa y una latencia promedio de 2-3 segundos.
- **Soporte Multilingüe:** Excelente desempeño nativo en español.

1.4. Estrategia de Recuperación Híbrida

Se implementó un pipeline de búsqueda avanzado que combina tres técnicas para maximizar la relevancia:

1. **Búsqueda Semántica:** Utilizando ChromaDB.
2. **Búsqueda por Keywords:** Utilizando el algoritmo BM25.
3. **Re-ranking:** Aplicación de un Cross-Encoder para refinar el orden de los resultados. Esta estrategia mejora significativamente la diversidad de documentos relevantes y reduce los falsos positivos inherentes a la búsqueda semántica pura.

---

2. Sistema Agéntico Multi-Herramienta (RAG Agent)

Esta sección describe la implementación del **Notebook 2**, donde se despliega un agente autónomo capaz de orquestar múltiples fuentes de datos.

2.1. Arquitectura del Agente

Se desarrolló un Agente basado en **LangChain** equipado con cuatro herramientas especializadas para cubrir distintos dominios de información:

1. **`doc_search_tool`:** Ejecuta búsquedas en documentación no estructurada (manuales, FAQs, tickets, reseñas) mediante el pipeline híbrido.
2. **`table_search_tool`:** Gestiona consultas sobre el catálogo estructurado. El LLM traduce lenguaje natural a operaciones de Pandas (e.g., "productos bajo $50k").
3. **`graph_search_tool`:** Resuelve consultas relacionales traduciendo la intención del usuario a consultas Cypher (e.g., "productos similares al P0001").
4. **`analytics_tool`:** Realiza análisis de ventas traduciendo consultas a SQL y generando visualizaciones automáticas profesionales con la librería **Seaborn**.

2.2. Validación y Resultados

El sistema fue sometido a pruebas en 6 categorías de consultas (técnicas, catálogo, relaciones, análisis, mixtas y casos borde), arrojando los siguientes resultados:

- **Precisión de Enrutamiento:** >95% en la selección de la herramienta adecuada.
- **Calidad de Respuesta:** Generación de respuestas contextuales con referencias específicas.
- **Robustez:** Manejo efectivo de ambigüedades en las consultas del usuario.

---

3. Trabajo Futuro y Mejoras Propuestas

Para optimizar el sistema y llevarlo a un entorno productivo, se proponen las siguientes líneas de acción:

1. **Actualización de Embeddings (Gemma):** Migrar de `all-MiniLM-L6-v2` a **Gemma** para mejorar el ratio costo/rendimiento y aumentar el *recall@5* en un 15-20%.
2. **Funciones Tabulares Avanzadas:** Incorporar filtros complejos (fecha de garantía, geolocalización) y comparaciones multi-atributo.
3. **Observabilidad (Tracing):** Integrar **LangSmith** para monitorear latencia, tasas de error y costos por consulta, permitiendo identificar cuellos de botella.
4. **Arquitectura Multi-Agente:** Evolucionar hacia un sistema de agentes especializados (Tabular, Grafos, Analytics) coordinados por un orquestador central para permitir paralelización.
5. **Persistencia SQL:** Migrar los datos históricos de CSV a **SQLite** para habilitar consultas más complejas, integridad referencial e índices optimizados.

---

4. Conclusiones

Logros Principales

- Desarrollo exitoso de un sistema RAG multi-modal con una precisión superior al 90%, integrando bases de datos vectoriales, tabulares y de grafos.
- Implementación de un pipeline híbrido (Semántica + Keywords + Re-ranking) que garantiza una recuperación robusta.

- Creación de un agente conversacional capaz de generar código SQL y Cypher válido (>95% de éxito) y visualizaciones de datos automáticas.

Justificación de Decisiones Técnicas

- **ChromaDB & Memgraph:** El equilibrio ideal entre practicidad para prototipado y potencia para consultas relacionales.
- **Clasificador LLM:** La inversión en costos de token se justifica plenamente por el aumento del 18.5% en la precisión frente al SVM.
- **Gemini 2.5 Flash:** Validado como el modelo más eficiente en su segmento de costo.

Lecciones Aprendidas

- Un esquema de base de datos bien documentado es crítico para la generación correcta de código Cypher por parte del LLM.
- El re-ranking es indispensable para asegurar la relevancia en la búsqueda vectorial.
- Los *System Prompts* detallados son la clave para reducir la ambigüedad en la selección de herramientas por parte del agente.

---

5. Bibliografía y Referencias

1. **LangChain Documentation - Agents:** https://docs.langchain.com/oss/python/langchain/agents
2. **Neo4j & Cypher Documentation:** https://neo4j.com/docs/

Anexo: Métricas de Evaluación

Desempeño del Clasificador de Intención:

- **Accuracy Global:** 88.9%
- **F1-Score por Categoría:**
    - Vectorial: 0.89
    - Tabular: 0.89
    - Grafo: 0.89

Desempeño de Herramientas del Agente:

- **Tasa de Éxito Global:** 96%
- **Detalle por Herramienta:**
    - `doc_search_tool`: 96% de éxito.