

Práctica de Laboratorio: Pipeline RAG Multi-Dominio con Scraping Concurrente y Aislamiento Vectorial

Asignatura: Sistemas Paralelos y Distribuidos

Modalidad: Grupal (máximo 2 integrantes) o Individual

Entorno de Ejecución: Ubuntu Linux / Docker Engine

1. Contexto y Justificación del Problema

Los sistemas de Generación Aumentada por Recuperación (RAG) empresariales requieren incorporar fuentes de conocimiento dinámicas (portales normativos, manuales web, circulares) sin degradar la precisión temática ni corromper las bases de conocimiento existentes.

Si múltiples dominios de información coexisten en una base vectorial sin segmentación, las consultas sufren **interferencia cruzada (cross-domain contamination)**: un término consultado sobre normativa bancaria puede recuperar erróneamente fragmentos de regulación tributaria o manuales técnicos debido a cercanía léxica en el espacio latente.

Asimismo, el ciclo de vida de ingesta (descarga HTTP, parsing HTML, tokenización/chunking, inferencia de embeddings e indexación en base de datos) combina operaciones de **I/O intensivo** (red/disco) con **cómputo intensivo** (procesamiento de texto y transformaciones matriciales). Desarrollar este pipeline de manera secuencial resulta ineficiente e inviable para corpus extensos.

2. Objetivo de la Práctica

Diseñar, implementar y evaluar un pipeline automatizado y concurrente en Python/FastAPI que:

1. Reciba una lista de URLs y una etiqueta de colección (coleccion_id / namespace).
2. Extraiga y preprocese el contenido textual en paralelo.
3. Genere embeddings vectoriales y los almacene en Supabase bajo un esquema de **aislamiento lógico multi-tenant**.
4. Permita consultas en tiempo real filtradas estrictamente por colección temática, midiendo cuantitativamente el rendimiento y la aceleración (*Speedup*) del procesamiento paralelo frente al secuencial.

3. Requerimientos Técnicos y Funcionales

Requerimiento 1: Aislamiento Vectorial en Base de Datos (Multi-Tenant)

- Modificar la tabla normativa_bancaria en PostgreSQL/Supabase incorporando una columna coleccion_id VARCHAR(50) indexada mediante B-Tree.
- Implementar o actualizar la función RPC match_normativa_coleccion en PL/pgSQL que evalúe la distancia coseno ($\leq$) **únicamente** sobre las tuplas que coincidan con la colección solicitada, garantizando cero contaminación cruzada entre especializaciones.

Requerimiento 2: Pipeline de Scraping e Ingesta Concurrente

- Implementar un script o endpoint /api/ingestar-web que acepte una carga útil JSON:

JSON

```
{
  "coleccion_id": "tributaria_bolivia_2026",
  "urls": [
    "https://sitio1.com/norma1",
    "https://sitio2.com/norma2"
  ],
}
```

```
"chunk_size": 1000,  
"chunk_overlap": 200,  
"concurrency_workers": 4  
}
```

- La descarga HTTP y limpieza de HTML debe desacoplarse utilizando técnicas de concurrencia:
- **Opción A (Multithreading/Async):** `asyncio` + `httpx` / `aiohttp`.
- **Opción B (Pool de Hilos/Procesos):** `concurrent.futures.ThreadPoolExecutor` para descarga de URLs y `ProcessPoolExecutor` para el particionamiento de texto (*chunking*).
- El envío de vectores al modelo de embeddings (`models/gemini-embedding-001`) debe realizarse por lotes (*batching*) gestionando de forma no bloqueante las pausas por tasa de peticiones (Rate Limit / Error 429).

Requerimiento 3: Consulta Especializada con Inferencia Blindada

- Adaptar el endpoint `/api/consultar` para recibir `coleccion_id` como parámetro obligatorio.
- La consulta vectorial debe restringirse a ese espacio. El prompt del LLM mantendrá la restricción determinista (`temperature=0.0`) y la regla de no-alucinación: si la colección no contiene antecedentes, el sistema debe declararlo explícitamente.

4. Métricas de Rendimiento Paralelo Requeridas

El reporte técnico debe incluir un análisis experimental riguroso comparando la ingesta secuencial frente a la concurrente sobre un conjunto de prueba mínimo de **10 URLs** de tamaño similar:

1. Tiempo Total de Ejecución (T_s vs. T_p):

- T_s : Tiempo de ingesta secuencial (1 worker / ejecución en bucle simple).
- T_p : Tiempo de ingesta paralela variando $p \in \{2, 4, 8\}$ workers.

2. Aceleración (Speedup, S_p):

$$S_p = \frac{T_1}{T_p}$$

Donde T_1 es el tiempo con un solo worker y T_p con p workers.

3. Eficiencia (E_p):

$$E_p = \frac{S_p}{p}$$

4. Identificación de Cuellos de Botella:

- Graficar el desglose temporal porcentual: % Tiempo Scraping/Red, % Tiempo Chunking/CPU, % Tiempo API Embeddings y % Tiempo Inserción Supabase.
- Justificar si la aceleración está limitada por la Ley de Amdahl debido al cuello de botella de I/O externo (API rate limits).

5. Criterios de Evaluación y Rúbrica (100 Puntos)

Criterio	Descripción Técnica	Puntaje
Aislamiento de Dominios en Base de Datos	Modificación del DDL, creación correcta de índices y función RPC PL/pgSQL que aísla de forma determinista las búsquedas por <code>coleccion_id</code> sin mezclar vectores. Demostración de cero contaminación cruzada en pruebas de consulta.	25 pts

Criterio	Descripción Técnica	Puntaje
Implementación Concurrente del Pipeline	Uso correcto de primitivas de paralelismo (asyncio, ThreadPoolExecutor o multiprocessing). Manejo adecuado de sincronización, ausencia de condiciones de carrera (<i>race conditions</i>) y control de excepciones en URLs no disponibles o lentas.	25 pts
Gestión de Lotes y Control de Rate Limiting	Implementación robusta de <i>batching</i> para generación de embeddings e inserciones masivas en base de datos. Control de flujo y reintentos exponenciales/pausas ante límites de cuota (429 / RESOURCE_EXHAUSTED).	20 pts
Análisis Experimental y Métricas de Rendimiento	Presentación de datos empíricos (T_s, T_p, S_p, E_p), tablas comparativas variando la cantidad de workers e interpretación analítica del comportamiento concurrente bajo la Ley de Amdahl.	20 pts
Contenedorización y Reproducibilidad	Dockerfile y docker-compose.yml funcionales que levantan el entorno completo (backend, scraper y base conectada) sin fallos de configuración de dependencias de sistema.	10 pts
Total		100 pts

6. Entregables

- Código Fuente:** Enlace a repositorio Git con historial de commits limpio, ramas de trabajo y el archivo .env.example.
- Entorno Docker:** Archivos Dockerfile y docker-compose.yml probados que permitan reproducir la práctica con un solo comando (docker compose up --build).

3. Informe de Rendimiento (PDF, máx. 5 páginas):

- Diagrama arquitectónico del flujo concurrente.
- Explicación de la función SQL/RPC desarrollada.
- Tablas y gráficos de Speedup y Eficiencia.
- Conclusiones técnicas sobre el límite de escalabilidad paralela frente a APIs de terceros.