Documentación Técnica

Generación de Embeddings y Almacenamiento Vectorial

10 de noviembre de 2025

Resumen

Este documento describe la arquitectura técnica seleccionada para la fase de vectorización del sistema RAG (Retrieval-Augmented Generation) del proyecto TEA Andalucía. Se detallan las decisiones tomadas respecto al modelo de embeddings, la base de datos vectorial y el flujo de procesamiento de datos desde los fragmentos de texto (chunks) hasta su almacenamiento indexado.

1. Esquema de la Solución

El subsistema de vectorización actúa como puente entre el procesamiento de texto y la futura interfaz de consulta. Su función es transformar los fragmentos de texto limpio en representaciones matemáticas (vectores) que permitan realizar búsquedas semánticas de alta precisión.

El flujo de datos implementado es el siguiente:

1. **Entrada:** Archivo `chunks_tea_andalucia.jsonl` conteniendo los fragmentos de texto pre-procesados y sus metadatos.
2. **Modelado:** Cada fragmento se envía al modelo de embeddings seleccionado para obtener su representación vectorial.
3. **Indexación:** Los vectores resultantes, junto con sus textos originales y metadatos, se insertan en la base de datos vectorial.
4. **Persistencia:** El índice completo se guarda en disco local para su reutilización sin necesidad de re-computar.

2. Elecciones Técnicas

2.1. Modelo de Embeddings: Google Gemini

Se ha seleccionado el modelo `models/text-embedding-004` de Google a través de la integración `GoogleGenerativeAIEmbeddings` de LangChain.

Justificación:

- **Rendimiento en Español:** Ofrece una excelente comprensión semántica en múltiples idiomas, incluido el español, crucial para interpretar correctamente la terminología legal y administrativa.
- **Eficiencia:** Proporciona un balance óptimo entre calidad de representación y velocidad de inferencia/coste.
- **Integración:** Su compatibilidad nativa con el ecosistema de Google y LangChain simplifica la implementación.

2.2. Base de Datos Vectorial: FAISS

Para el almacenamiento y recuperación de vectores se utiliza **FAISS (Facebook AI Similarity Search)**.

Justificación:

- **Alto Rendimiento:** Es una de las bibliotecas más rápidas y eficientes para búsqueda de similitud en grandes conjuntos de vectores densos.
- **Simplicidad Local:** No requiere una infraestructura de servidor compleja (como bases de datos externas), funcionando perfectamente como una solución embebida que persiste en archivos locales.
- **Flexibilidad:** Permite la actualización del índice (añadir nuevos documentos) y su fácil recarga para consultas posteriores.

3. Implementación del Flujo

El script `vectorizer_from_chunks.py` orquesta todo el proceso. Utiliza un enfoque por lotes (batch processing) para optimizar las llamadas a la API de embeddings y la inserción en FAISS, gestionando eficientemente la memoria y los límites de tasa de la API.

```
# Procesamiento por lotes para eficiencia
batch_size = 50
remaining_docs = documents[batch_size:]

if remaining_docs:
    with tqdm(total=len(remaining_docs), desc="Vectorizando") as pbar:
        for i in range(0, len(remaining_docs), batch_size):
            batch = remaining_docs[i:i + batch_size]
            # Genera embeddings y añade al indice
            self.vectorstore.add_documents(batch)
            pbar.update(len(batch))

# Persistencia del indice en disco
self.vectorstore.save_local(index_path)
```

Listing 1: Fragmento clave de la vectorización por lotes

4. Conclusión

La combinación de **Google Gemini** para embeddings y **FAISS** para el almacenamiento vectorial proporciona una base sólida, escalable y de alto rendimiento para las capacidades de búsqueda semántica del proyecto, cumpliendo con todos los requisitos funcionales de actualización y precisión en español.