RAG + LM Studio + Postgres/pgvector — README

Este proyecto implementa un **pipeline RAG local**: ingesta de documentos → embeddings → almacenamiento en Postgres con **pgvector** → recuperación semántica → generación de respuestas con un **LLM servido por LM Studio**. Todo corre en tu VM de Ubuntu (cliente) y un host con LM Studio en LAN.

Índice

---

1) Arquitectura (visión general)

```
[data/*.txt|.md|.pdf]
        │
        ▼
  ingest.py     → troceado (chunking) + embeddings (LM Studio o ST)
        │
        ▼
Postgres + pgvector  → tabla `documents(… embedding VECTOR(d))`
        ▲
        │   query.py → embebe la consulta, recupera top-k por similitud (cosine)
        │
```

```
        └── llm_client.py → LM Studio (/v1/chat/completions) para redactar la
respuesta
```

- **Ingesta:** trocea tus documentos y calcula **embeddings** de cada fragmento; los guarda en Postgres con índice vectorial (**HNSW** + cosine).
- **Consulta:** convierte tu pregunta en embedding, recupera los **k** pasajes más similares y arma un **prompt con contexto** para que el LLM responda **anclado en tus fuentes**.
- **LM Studio:** servidor **OpenAI-compatible** en tu LAN (p. ej., `http://192.168.0.194:1234/v1`).

---

2) Requisitos

- Ubuntu 22.04/24.04 con Python 3.11 (venv), Node opcional, Docker.
- Contenedor Postgres con extensión **pgvector**.
- **LM Studio** ejecutándose en el host (modo Developer → Local Server).

---

3) Estructura del proyecto

```
ai-stack/
├─ rag/
│  ├─ .venv/                # entorno virtual de Python
│  ├─ ingest.py             # ingesta: troceo + embeddings + upsert
│  ├─ query.py              # consulta: retrieve + prompt + LLM
│  ├─ llm_client.py         # cliente de chat LM Studio (OpenAI-compatible)
│  ├─ .env                  # configuración central del proyecto
│  └─ data/                 # tus documentos (puede cambiarse con DATA_DIR)
├─ infra/
│  └─ docker-compose.yml    # Postgres + pgvector
└─ mcp/, agent/             # (reservados para siguientes fases)
```

---

4) Configuración (.env)

El fichero `rag/.env` es **el panel de mandos**. Variables clave:

```
# — LLM de chat (LM Studio) —
LLM_BASE_URL=http://192.168.0.194:1234/v1
LLM_API_KEY=lm-studio                 # cadena cualquiera
LLM_MODEL=llama-3.2-3b-instruct-uncensored  # elige un modelo de chat de /v1/
models

# — Base de datos —
DATABASE_URL=postgresql+psycopg://postgres:postgres@localhost:5432/ragdb
```

```
# — Documentos —
DATA_DIR=/home/ayax/ai-stack/rag/data     # carpeta con .txt .md .pdf
(recursivo)
CHUNK_SIZE=800
CHUNK_OVERLAP=120

# — Embeddings — (opción A: LM Studio)
EMBEDDING_BACKEND=lmstudio
EMBEDDING_BASE_URL=http://192.168.0.194:1234/v1
EMBEDDING_API_KEY=lm-studio
EMBEDDING_MODEL=text-embedding-embeddinggemma-300m
# EMBEDDING_ENDPOINT=embeddings          # opcional; fallback automático a
'embedding'
# EMBEDDING_DIM=768                      # opcional para evitar probe inicial

# — Embeddings — (opción B: locales con Sentence-Transformers)
# EMBEDDING_BACKEND=sentence-transformers
# EMBEDDING_MODEL=BAAI/bge-m3
```

> **Consejo:** comprueba los modelos cargados con: `curl http://192.168.0.194:1234/v1/models` y pon en `LLM_MODEL` / `EMBEDDING_MODEL` el **id exacto** que te devuelva.

---

5) Puesta en marcha

5.1 Base de datos

```
cd ~/ai-stack/infra
docker compose up -d
```

5.2 LM Studio

1. Abre **Developer → Local Server**.
2. Selecciona un **modelo de chat** y/o un **modelo de embeddings** (pueden convivir).
3. Puerto: `1234` · Bind: `0.0.0.0` · **Run**.
4. Verifica desde la VM:

```
curl http://192.168.0.194:1234/v1/models
curl -s http://192.168.0.194:1234/v1/embeddings
  -H "Authorization: Bearer lm-studio" -H "Content-Type: application/
json"
  -d '{"model":"text-embedding-embeddinggemma-300m","input":"hola"}' |
jq '.data[0].embedding | length'
```

5.3 Dependencias Python (solo una vez)

```
cd ~/ai-stack/rag
python -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
pip install openai httpx python-dotenv sqlalchemy "psycopg[binary]"
pgvector
            langchain pypdf sentence-transformers numpy
```

5.4 Ingesta de documentos

Pon tus ficheros en `DATA_DIR` (por defecto `rag/data/`) y ejecuta:

```
cd ~/ai-stack/rag && source .venv/bin/activate
python ingest.py
```

Salida esperada:

```
📂 Usando DATA_DIR: /home/ayax/ai-stack/rag/data
🔎 Embeddings LM Studio → http://192.168.0.194:1234/v1/embeddings |
model=text-embedding-embeddinggemma-300m
🔗 Ingesta completada.
```

5.5 Consulta

```
python query.py "¿Qué es MCP?"
```

La respuesta citará fragmentos en el formato `[doc#chunk]`.

---

6) ¿Cómo funciona por dentro?

6.1 `ingest.py`

- **Lee** variables de `.env` (DB, DATA_DIR, chunking, backend de embeddings).
- **Trocea** cada documento con `RecursiveCharacterTextSplitter(CHUNK_SIZE, CHUNK_OVERLAP)`.
- **Embeddings** (a elección):
- **LM Studio**: llamada HTTP directa (con `httpx`) a `/v1/embeddings` con `EMBEDDING_MODEL`.
- **Sentence-Transformers**: modelo local (normaliza para cosine).
- **Schema**: crea `documents` con `embedding VECTOR(d)` e índice `HNSW (vector_cosine_ops)` si no existen.
- **Inserta** filas (`doc_id`, `chunk_id`, `content`, `metadata`, `embedding`).

6.2 `query.py`

- **Embebe** la pregunta con **el mismo backend** de embeddings.
- **Consulta** por similitud:

```
SELECT ..., 1 - (embedding <=> :qvec) AS score
FROM documents
ORDER BY embedding <=> :qvec
LIMIT :k;
```

- **Contexto**: concatena los `k` pasajes más similares.
- **Generación**: envía `system + user` a `llm_client.py`, que usa LM Studio `/v1/chat/completions`.

6.3 `llm_client.py`

- Cliente **OpenAI-compatible** apuntando a `LLM_BASE_URL`.
- Parámetros útiles: `temperature`, mensajes de `system` y `user`.

---

7) Cambiar de modelo de embeddings (dimensión)

pgvector exige una **dimensión fija** por columna. Si cambias `EMBEDDING_MODEL` y la dimensión no coincide con la ya almacenada:

```
docker exec -it infra-pgvector-1 psql -U postgres -d ragdb -c "DROP TABLE documents;"
python ingest.py
```

> Si prefieres evitar el *probe* de dimensión en LM Studio, define `EMBEDDING_DIM` en `.env` (p. ej., 768).

---

8) Comandos de verificación útiles

```
# Red/puerto abierto
nc -vz 192.168.0.194 1234

# Modelos disponibles
curl -s http://192.168.0.194:1234/v1/models | jq

# Probar embeddings (y ver dimensión)
curl -s http://192.168.0.194:1234/v1/embeddings -H "Authorization: Bearer lm-studio"
  -H "Content-Type: application/json"
  -d '{"model":"text-embedding-embeddinggemma-300m","input":"hola"}' | jq '.data[0].embedding | length'
```

```
# Estado del contenedor Postgres
cd ~/ai-stack/infra && docker compose ps
```

---

9) Solución de problemas (FAQ)

- **Timeout/Connection error en embeddings**
- Comprueba `curl …/v1/models` y `…/v1/embeddings` desde la VM.
- En LM Studio, usa **Bind** `0.0.0.0` y abre el puerto en el firewall.
- Verifica que `EMBEDDING_MODEL` exista exactamente en `/v1/models`.
- `model not found`
- El `id` en `.env` no coincide con el listado de `/v1/models`.
- **Error de dimensión**
- Cambiaste de modelo de embeddings → borra/migra `documents` y reingesta.
- **Respuestas vagas / alucinaciones**
- Baja `temperature` en `llm_client.py` (0.1–0.2) y endurece el *system prompt*.
- Ajusta `k` y `CHUNK_SIZE`/`CHUNK_OVERLAP` para mejorar el contexto relevante.
- **Lentitud**
- Asegura índice HNSW creado; reduce `k` o el tamaño de chunk; usa un modelo de embeddings más ligero.

---

10) Extensiones recomendadas

- **Evaluación de RAG**: integra RAGAS/TruLens para medir *groundedness*, relevancia y cobertura.
- **Reranking**: añade un reranker (cross-encoder) tras el retriever para mejorar precisión.
- **Caching**: cachea embeddings y/o respuestas recurrentes.
- **MCP + agentes**: expón tu vector DB como **resource** y añade **tools** para reindexar/evaluar.

---

11) Notas de seguridad

- LM Studio no valida realmente el API key → **no** expongas el puerto fuera de tu LAN.
- Limita acceso por firewall a la IP de la VM y mantén credenciales de Postgres fuera del código.

---

12) Licencia / créditos

Proyecto de aprendizaje personal. Usa componentes open-source y LM Studio como servidor local OpenAI-compatible.