> **LO ESENCIAL · Módulo 6 · RAG que funciona**
>
> La calidad de recuperación pesa más que el tamaño del modelo. Permisos antes de recuperar.

6.1 · Ingesta y chunking

> Objetivos
>
> - Cortar por estructura, no por tamaño
> - Ingerir 50 documentos reales con metadatos

Lo que entra mal, sale mal

Un RAG es tan bueno como sus fichas. Si los documentos se recortan por tamaño («cada 500 caracteres»), una tabla queda partida en dos, un artículo de contrato pierde su número, y una factura queda sin su encabezado. Después ningún modelo lo arregla.

Regla: se corta por **estructura**, y cada ficha lleva **metadatos** suficientes para citarla.

El flujo de ingesta

Chunking por estructura

Tipo de documento	Unidad de corte	Tamaño típico	Solapamiento
Contrato	cláusula / artículo numerado	200–600 tokens	0 (las cláusulas son unidades)
Factura	**una ficha por factura** (encabezado + líneas + totales)	300–900	0
Acta	punto del orden del día	150–500	1 frase
Manual / política	encabezado de sección (##)	300–800	10 %
Correo	un correo	100–800	0
Texto sin estructura	párrafos, agrupados hasta el tope	400–600	10–15 %

Cada ficha lleva:

```
{"id":"contrato-2026-014#cl-7","texto…":"","origen":"contratos/2026-014.pdf","pagina":3,
 "seccion":"Cláusula 7 — Garantía","coleccion":"contratos","permisos":["gerencia","contabilidad
    "],
 "fecha_doc":"2026-03-12","hash":"sha256…:"}
```

El campo `permisos` es el que el módulo 6.3 usa para filtrar **antes** de buscar. Se pone en la ingesta, no después.

> **PROCEDIMIENTO · Laboratorio 6.1 · ingerir 50 documentos reales**

```
cd ~/labs/el-tornillo && mkdir -p labs/06-01 && cd labs/06-01 && pip install -q pymupdf
```

```
# ingerir.py — PDF con capa de texto → fichas por estructura con metadatos
import fitz, json, re, hashlib, pathlib, sys
def fichas_contrato(texto, origen):
    partes = re.split(r"\n(?=(?:CL[ÁA]USULA|ART[ÍI]CULO)\s+\w+)", texto)
    for i, p in enumerate(partes):
        if len(p.strip()) < 40: continue
        titulo = p.strip().split("\n")[0][:80]
        yield {"id": f"{pathlib.Path(origen).stem}#{i}", "texto": p.strip(), "origen":
    origen,
               "seccion": titulo, "coleccion": "contratos", "permisos": ["gerencia","
    contabilidad"],
               "hash": "sha256:"+hashlib.sha256(p.encode()).hexdigest()[:16]}
salida = []
for pdf in sys.argv[1:]:
    doc = fitz.open(pdf)
    texto = "\n".join(pag.get_text() for pag in doc)
    if len(texto.strip()) < 200: print("SIN CAPA DE TEXTO → OCR:", pdf, file=sys.stderr);
    continue
    salida += list(fichas_contrato(texto, pdf))
json.dump(salida, open("fichas.json","w"), ensure_ascii=False, indent=1)
print(len(salida), "fichas")
```

Lo que debe ver: N fichas, cada una con su cláusula, y **una lista de PDFs sin capa de texto** que van al módulo 7. Revise a mano diez fichas: ¿la cláusula está completa? ¿el título es el correcto?

Entregable: `ingerir.py`, `fichas.json`, y `labs/06-01/revision.md` con las diez fichas revisadas. Commit: «Capa 6.1: ingesta».

Nada en la ingesta con capa de texto (CPU). Los sin capa dependen del módulo 7.

Cuándo NO usar esto **Ingesta y chunking**

- No use chunking fijo «porque es lo que trae la biblioteca». Es la causa más común de RAG malo.
- No ingiera todo el archivo histórico el primer día: 50 documentos representativos, medir, luego el resto.

PRÁCTICA · Práctica

1. Escriba la función `fichas_factura`: una ficha por factura, con líneas y totales juntos.
2. Un acta de 12 páginas con 9 puntos: ¿cuántas fichas? ¿Qué lleva cada una en `seccion`?

Referencias

1. PyMuPDF — documentación: https://pymupdf.readthedocs.io/
2. Docling — extracción con estructura de PDF/DOCX: https://github.com/docling-project/docling `[VOLÁTIL — verificado: 2026-09]`
3. Lewis, P. et al. (2020). RAG — arXiv:2005.11401.

6.2 · Embeddings y recuperación

> Objetivos
>
> - Elegir embedding por idioma y dominio
> - Medir recall@k con 40 consultas

Elegir el modelo de embeddings por criterio

El modelo de embeddings decide qué fichas aparecen. Es pequeño (cientos de MB), corre en CPU con soltura, y **se cambia con menos dolor que el modelo de lenguaje** —siempre que se reconstruya el índice.

Criterios `[VOLÁTIL — verificado: 2026-09]`:

Criterio	Por qué	Cómo se verifica
Español de verdad, no «multilingüe» de etiqueta	los documentos son en español con jerga contable	recall@5 en **sus** 40 consultas
Dimensión 384–1 024	más dimensiones ≠ mejor; más memoria en el índice	tamaño del índice
Longitud máxima ≥ 512 tokens	fichas de contrato son largas	ficha técnica del modelo
Licencia permisiva	va a producción en un cliente	la licencia del modelo
Dos alternativas por tamaño	pequeño (~100 MB) para A; medio (~500 MB) para B/C	medir las dos

El tablero público **MTEB** ordena modelos por tareas; sirve para preseleccionar tres, no para

elegir: la elección la hacen sus consultas.

Índice

Para una PyME, un **índice en la misma base de datos** que el resto (PostgreSQL con `pgvector`) es suficiente hasta cientos de miles de fichas, y evita otro servicio que mantener. Un motor vectorial dedicado se justifica con millones de fichas o latencias de milisegundos.

```
CREATE EXTENSION IF NOT EXISTS vector;
CREATE TABLE fichas (id text PRIMARY KEY, texto text, origen text, seccion text,
  coleccion text, permisos text[], fecha_doc date, emb vector(768));
CREATE INDEX ON fichas USING hnsw (emb vector_cosine_ops);
CREATE INDEX ON fichas USING gin (permisos);
```

> **PROCEDIMIENTO · Laboratorio 6.2 · recall@k con 40 consultas**
>
> ```
> # evaluar_recuperacion.py — ¿la ficha correcta aparece entre las k primeras?
> import json, psycopg, numpy as np
> from sentence_transformers import SentenceTransformer
> m = SentenceTransformer("<modelo-embeddings>")            # [VOLÁTIL] probar dos
> consultas = json.load(open("consultas.json"))              # [{"q": "...", "ficha_esperada
>     ": "contrato-2026-014#7"}, ...]
> con = psycopg.connect("dbname=tornillo")
> def recall(k):
>     aciertos = 0
>     for c in consultas:
>         v = m.encode(c["q"], normalize_embeddings=True).tolist()
>         ids = [r[0] for r in con.execute("SELECT id FROM fichas ORDER BY emb <=> %s::
>     vector LIMIT %s", (v, k))]
>         aciertos += c["ficha_esperada"] in ids
>     return aciertos/len(consultas)
> for k in (1, 3, 5, 8): print(f"recall@{k}: {recall(k):.2f}")
> ```
>
> Lo que debe ver: cuatro cifras por modelo. Un recall@5 por debajo de 0,8 en sus consultas significa que el RAG va a responder con las fichas equivocadas una de cada cinco veces, y **ningún modelo de lenguaje lo corrige**.
> **Entregable:** `consultas.json` (40, escritas con el cliente), y la tabla recall@k de dos modelos. Commit: «Capa 6.2: recuperación medida».

Qué se degrada en cada perfil

A — Embeddings en CPU: ingerir 1 000 fichas tarda minutos, no segundos. Modelo pequeño

B — Modelo medio; reindexar es rápido

Cuándo NO usar esto **Embeddings y recuperación**

- No use embeddings para **búsqueda exacta** (número de factura, cédula): eso es `WHERE numero = …`.
- No mida recall con consultas inventadas por usted: las escribe el cliente, con sus palabras.
- No cambie de modelo de embeddings sin reindexar todo: los vectores no son compatibles.

PRÁCTICA · **Práctica**

1. Combine búsqueda exacta (`ILIKE` sobre `seccion`) con vectorial. ¿Cuándo gana cada una?
2. Su recall@5 es 0,72. Liste tres causas posibles en orden de probabilidad.

Referencias

1. pgvector — https://github.com/pgvector/pgvector
2. MTEB — tablero: https://huggingface.co/spaces/mteb/leaderboard `[VOLÁTIL — verificado: 2026-09]`
3. Reimers, N. y Gurevych, I. (2019). Sentence-BERT — arXiv:1908.10084.

6.3 · Reranking y filtrado por permisos ANTES de recuperar

> Objetivos
>
> - Poner el permiso como filtro del índice
> - Añadir reranking y medir la ganancia

El permiso es un filtro del índice, no un post-proceso

El error grave: recuperar las 8 mejores fichas y **después** quitar las que el usuario no puede ver. Dos problemas: (1) al vendedor le pueden quedar 2 fichas de 8, y el modelo responde mal; (2) peor, si el filtro falla, **la nómina ya está en el contexto**. El permiso se aplica en la consulta:

```
-- el vendedor solo puede ver 'ventas' y 'publico'
```

```
SELECT id, texto, seccion, origen FROM fichas
WHERE permisos && %s::text[]                 -- grupos del usuario (de la cabecera del SSO, 3.2)
  AND coleccion = ANY(%s)                    -- colección pedida
ORDER BY emb <=> %s::vector LIMIT 20;
```

Los grupos vienen de la cabecera que puso el proxy (módulo 3.2). El asistente **no** los recibe del usuario ni del modelo.

Reranking

La búsqueda vectorial trae candidatos parecidos en forma. Un **reranker** —un modelo pequeño que lee la pregunta y cada candidato juntos— reordena por relevancia real y descarta los que no responden. Cuesta unos cientos de milisegundos por 20 candidatos en CPU.

```
from sentence_transformers import CrossEncoder
rr = CrossEncoder("<modelo-reranker-multilingue>")      # [VOLÁTIL — verificado: 2026-09]
candidatos = buscar(pregunta, grupos, k=20)              # con el filtro de permisos
puntajes = rr.predict([(pregunta, c["texto"]) for c in candidatos])
mejores = [c for c, p in sorted(zip(candidatos, puntajes), key=lambda x: -x[1]) if p > 0.2][:5]
```

> PROCEDIMIENTO · **Laboratorio 6.3 · medir la ganancia y probar el aislamiento**
>
> 1. Repita `evaluar_recuperacion.py` con reranking sobre 20 candidatos: recall@5 **antes y después**. Ganancia típica: 5–15 puntos. Si no gana, el reranker no es para su idioma.
> 2. Prueba de aislamiento: como `vendedor`, 20 preguntas sobre nómina y contratos. Resultado esperado: **cero** fichas de esas colecciones en el contexto, y una respuesta del tipo «no tengo acceso a esa información». Se registra en la traza (módulo 11).
>
> **Entregable:** tabla antes/después, y `labs/06-03/aislamiento.md` con las 20 pruebas. Commit: «Capa 6.3: permisos y reranking».

Qué se degrada en cada perfil

A — Reranking en CPU: +0,5–1,5 s por consulta. Aceptable en lote, justo en vivo

B — +100–300 ms; se puede rerankear 30–50 candidatos

Cuándo NO usar esto **Reranking y filtrado por permisos ANTES de recuperar**

- No rerankee 5 candidatos: no hay nada que reordenar. Recupere 20–50, rerankee, entregue 5.

- No use el reranker como filtro de permisos «porque puntúa bajo lo que no corresponde»: el permiso es determinista, el reranker no.

PRÁCTICA · **Práctica**

1. Un usuario está en `ventas` y `contabilidad`. Escriba la consulta SQL.
2. ¿Qué pasa con las fichas cuyo `permisos` está vacío? Decida y justifique.

Referencias

1. pgvector — filtrado combinado con HNSW: https://github.com/pgvector/pgvector
2. Nogueira, R. y Cho, K. (2019). *Passage Re-ranking with BERT.* arXiv:1901.04085.
3. OWASP LLM Top 10 — LLM06 (divulgación de información sensible).

6.4 · Diagnóstico de un RAG malo

> Objetivos
>
> - Localizar la falla: ingesta, recuperación o generación
> - Aplicar la lista de verificación a un RAG roto a propósito

Un RAG malo casi nunca falla donde parece

Cuando la respuesta es mala, el instinto es cambiar el modelo. En la mayoría de los casos el modelo es la última causa. El diagnóstico va **de atrás hacia adelante**:

Etapa	Pregunta	Cómo se comprueba	Falla típica
1 · Ingesta	¿la información está en alguna ficha, completa?	buscar a mano en `fichas.json`	ficha partida; PDF sin capa de texto ignorado; tabla destruida
2 · Recuperación	¿la ficha correcta aparece en los k candidatos?	recall@k (6.2)	embedding malo para español; consulta demasiado corta; filtro de permisos mal puesto
3 · Reranking	¿queda entre las 5 entregadas?	comparar antes/después (6.3)	umbral demasiado alto; reranker no multilingüe
4 · Contexto	¿cabe en la ventana, en buen orden?	presupuesto (5.1)	ficha correcta cortada por el tope; pregunta al principio
5 · Generación	con la ficha correcta en contexto, ¿responde bien?	pasar la ficha a mano	modelo demasiado cuantizado; instrucciones ambiguas; sin esquema

Solo si las etapas 1–4 pasan y la 5 falla, el problema es el modelo.

> PROCEDIMIENTO · **Laboratorio 6.4 · un RAG roto a propósito**
>
> Se entregan 5 defectos plantados en `labs/06-04/rag-roto/` (chunking fijo, un PDF sin OCR, filtro de permisos invertido, umbral de reranking en 0,9, pregunta al inicio del prompt). Con la lista de arriba, localice cada uno y documente: síntoma → etapa → prueba que lo confirmó → arreglo.
> **Entregable:** `labs/06-04/diagnostico.md` con los 5. Commit: «Capa 6: RAG con diagnóstico».
> Nada: el diagnóstico es método. En A las pruebas tardan más.

Cuándo NO usar esto **Diagnóstico de un RAG malo**

- No arranque por la etapa 5. Nunca.
- No cambie dos cosas a la vez: no sabrá cuál arregló (o rompió).

PRÁCTICA · **Práctica**

1. «El asistente dice que el contrato no tiene cláusula de garantía, y sí la tiene.» Recorra la tabla y diga la etapa más probable.
2. Diseñe una consulta de canario (módulo 12) que detecte la etapa 2 rota cada noche.

Referencias

1. Módulos 5, 6.1–6.3 y 12 de este curso.
2. Barnett, S. et al. (2024). *Seven Failure Points When Engineering a Retrieval Augmented Generation System*. arXiv:2401.05856.