PERTEMUAN 10 · PRAKTIK

RAG Part 1: *Embedding & Vector Data*

Membangun separuh pertama sistem RAG: me
indeks yang bisa dicari berdasarkan makna. Ba

Praktik | 100 menit | CPMK: C3, C4

Tujuan Pembelajaran

Di P-08 kita melihat bahwa LLM bisa berhalusinasi dan tidak tahu dokumen internal. Di P-09 kita belajar memanggil LLM dari kode. Hari ini kita membangun komponen yang memberi LLM *bahan* untuk menjawab. Setelah sesi ini, mahasiswa diharapkan mampu:

1. Menjelaskan masalah yang diselesaikan RAG dan membandingkannya dengan fine-tuning serta long context
2. Menguraikan dua alur RAG: *indexing* (offline) dan *query* (saat pertanyaan datang)
3. Menjelaskan embedding dan cosine similarity sebagai dasar pencarian semantik
4. Memilih ukuran chunk dan overlap dengan mempertimbangkan kompromi presisi dan konteks
5. Membangun indeks dokumen di vector database (ChromaDB) dan melakukan pencarian
6. Mengevaluasi kualitas retrieval secara terukur sebelum menambahkan LLM

🛠️ **Persiapan:** Python 3.9+, 2–3 file PDF berbasis teks (bukan hasil scan) untuk dijadikan bahan uji, dan koneksi internet untuk mengunduh model embedding saat pertama kali

1. Mengapa RAG?

RAG (*Retrieval-Augmented Generation*) adalah pola di mana sistem lebih dulu *mencari* potongan dokumen yang relevan, lalu menyertakannya di prompt agar LLM menjawab berdasarkan potongan itu. Ia menjawab tiga keterbatasan LLM dari P-08: halusinasi, pengetahuan yang usang, dan ketidaktahuan terhadap data privat.

PENDEKATAN	CARA KERJA	KELEBIHAN	KETERBATASAN
RAG	Cari dokumen relevan, sisipkan ke prompt	Mudah diperbarui (ganti dokumen), jawaban bisa menyebut sumber, tidak perlu melatih model	Kualitas bergantung pada retrieval; menambah komponen yang harus dikelola
Fine-tuning	Melatih lanjut model pada data Anda	Cocok untuk gaya, format, atau perilaku tertentu	Mahal, sulit diperbarui, kurang cocok untuk menyuntikkan fakta yang sering berubah
Long context	Masukkan seluruh dokumen ke prompt	Sederhana untuk dokumen pendek	Biaya token tinggi, terbatas context window, relevansi bisa tenggelam di antara teks panjang

🎯 **Kapan RAG tepat?** Saat jawaban harus berakar pada kumpulan dokumen yang besar, sering berubah, atau privat: pedoman akademik, SOP perusahaan, katalog produk, kontrak. Inilah skenario **Proyek 4 (Chatbot Akademik RAG)**.

2. Dua Alur dalam Sistem RAG

ALUR A · INDEXING (OFFLINE) — FOKUS HARI INI

A1	→	A2	→	A3	→	A4
Load		**Split**		**Embed**		**Store**
Baca dokumen (PDF, DOCX, web) menjadi teks		Potong teks menjadi chunk berukuran terkelola		Ubah tiap chunk menjadi vektor angka		Simpan vektor + teks + metadata di vector database

ALUR B · QUERY (SAAT PERTANYAAN DATANG) — LANGKAH B1–B2 HARI INI, B3–B4 DI P-11

B1	→	B2	→	B3 · P-11	→	B4 · P-11
Embed pertanyaan		**Retrieve top-k**		**Susun prompt**		**Generate**
Pakai model embedding yang sama dengan A3		Ambil k chunk dengan vektor paling dekat		Pertanyaan + chunk + instruksi "jawab hanya dari konteks"		LLM menjawab, idealnya dengan menyebut sumber

⚠️ **Aturan emas:** model embedding untuk dokumen (A3) dan untuk pertanyaan (B1) harus *sama*. Vektor dari model berbeda hidup di "ruang" yang berbeda dan tidak bisa dibandingkan.

3. Embedding & Pencarian Semantik

Embedding adalah representasi teks sebagai vektor (daftar angka, biasanya ratusan dimensi) yang dilatih agar teks bermakna mirip memiliki vektor yang berdekatan. Dua kalimat yang kata-katanya berbeda tetapi artinya sama, misalnya *"cuti akademik"* dan *"berhenti kuliah sementara"*, seharusnya berdekatan di ruang vektor. Ini berbeda dari pencarian kata kunci yang hanya mencocokkan kata persis.

Mengukur kedekatan: cosine similarity

$$\cos(A, B) = (A \cdot B) / (\|A\| \cdot \|B\|)$$

berarti lebih mirip.

Coba sendiri: mini search engine

Di bawah ini ada 8 potongan "peraturan akademik" (**contoh fiktif**, bukan peraturan UKWMS). Pencarian ini memakai **TF-IDF + cosine similarity** (berbasis kata) agar bisa dijalankan di browser. Ini bukan embedding neural, tetapi mekanisme top-k dan cosine-nya sama. Coba tiga contoh pencarian dan bandingkan hasilnya.

Ketik pertanyaan, mis. syarat cuti akademik

Coba: ① `syarat cuti akademik` ② `boleh berhenti kuliah sementara?`

③ `minimal kehadiran agar boleh ujian`

Ketik pertanyaan atau klik salah satu contoh di atas.

🔍 **Apa yang terlihat?** Contoh ① menemukan peraturan cuti karena kata *cuti* dan *akademik* persis sama. Pada contoh ②, pertanyaannya bermakna sama tetapi tidak berbagi kata kunci dengan peraturan cuti, sehingga yang muncul justru potongan yang kebetulan memuat kata "kuliah" (soal nilai mata kuliah dan biaya kuliah) dengan skor sangat rendah, sementara peraturan cuti tidak masuk tiga teratas. Itulah keterbatasan pencarian berbasis kata. Model embedding neural dilatih menangkap makna sehingga "berhenti kuliah sementara" cenderung dekat dengan "cuti akademik". Meski begitu, hasilnya *tidak dijamin* sempurna, sehingga retrieval tetap harus dievaluasi (bagian 7).

🌐 **Pilih model embedding sesuai bahasa.** Model embedding bawaan banyak pustaka dilatih terutama pada bahasa Inggris. Untuk dokumen Bahasa Indonesia, gunakan model *multilingual*. Model embedding yang dijalankan lokal tidak memakai API sehingga tidak ada biaya per token, tetapi membutuhkan unduhan model dan memori yang cukup.

4. Chunking: Memotong Dokumen dengan Bijak

Dokumen tidak dimasukkan utuh ke indeks. Ia dipotong menjadi *chunk* karena (1) satu vektor untuk dokumen panjang merata-ratakan terlalu banyak topik sehingga kabur, dan (2) hanya potongan relevan yang sebaiknya masuk ke prompt agar hemat token dan fokus.

TERLALU KECIL

Konteks terpotong

- Kalimat jawaban terpisah dari penjelasannya
- Retrieval menemukan potongan yang "benar" tetapi tidak cukup untuk menjawab

TERLALU BESAR

Relevansi mengabur

- Satu chunk memuat banyak topik; skor kemiripan jadi kurang tajam
- Prompt penuh teks tak relevan, biaya naik

OVERLAP

Tumpang tindih sebagian

- Beberapa karakter akhir chunk diulang di awal chunk berikutnya
- Mengurangi risiko informasi penting terbelah di perbatasan, dengan harga indeks sedikit lebih besar

Coba sendiri: ukuran chunk & overlap

Teks contoh di bawah (fiktif) dipotong menurut dua pengaturan. Bagian yang ditandai adalah overlap dengan chunk sebelumnya.

Ukuran chunk 200

Overlap 30

Total teks 454 karakter → 3 chunk. 2 dari 3 chunk tidak berakhir di akhir kalimat.

```
CHUNK 1 · karakter 0–200 · 200 karakter
```

Selama cuti, mana

CHUNK 2 · karakter 170–370 · 200 karakter

lan dimulai. Selama cuti, mahasiswa tidak diperkenankan mengikuti kegiatan perkuliahan dan ujian. Masa cuti tidak dihitung sebagai bagian dari batas masa studi. Mahasiswa yang tidak melakukan registra

CHUNK 3 · karakter 340–454 · 114 karakter

yang tidak melakukan registrasi tanpa izin cuti dinyatakan mengundurkan diri setelah dua semester berturut-turut.

Strategi pemotongan

STRATEGI	CARA KERJA	CATATAN
Ukuran tetap (karakter/token)	Potong tiap N karakter, dengan overlap	Paling sederhana, tetapi bisa memotong di tengah kalimat atau tabel
Rekursif	Coba pisah di paragraf dulu, lalu kalimat, lalu kata, sampai muat dalam ukuran	Titik awal yang baik; menjaga batas alami teks
Berdasarkan struktur	Potong per bab, pasal, atau heading	Cocok untuk dokumen terstruktur seperti peraturan; simpan judul bagian sebagai metadata

🧪 **Tidak ada ukuran chunk yang selalu benar.** Rentang awal yang sering dipakai adalah beberapa ratus hingga sekitar seribu karakter dengan overlap 10–20%, tetapi angka ini hanya titik awal untuk *diuji* pada data Anda. Proyek 4 secara khusus meminta perbandingan ukuran chunk 500, 1.000, dan 1.500 karakter.

5. Vector Database

pendekatan (*approximate nearest neighbor*) agar tidak perlu membandingkan query dengan semua vektor satu per satu.

DISIMPAN

Tiga hal per chunk

- **Vektor** embedding (untuk pencarian)
- **Dokumen** teks asli chunk (untuk prompt)
- **Metadata** seperti nama file, nomor halaman, bab (untuk sitasi dan filter)

PILIHAN UMUM

ChromaDB, FAISS, dan lainnya

- **ChromaDB:** mudah dipakai, bisa disimpan lokal; cocok untuk belajar dan proyek kecil
- **FAISS:** pustaka pencarian vektor cepat, lebih "mentah"
- **Layanan terkelola / ekstensi database:** untuk skala produksi

Kita memakai **ChromaDB** karena instalasinya sederhana dan tidak butuh server terpisah. Konsep yang dipelajari (koleksi, add, query, metadata) berlaku serupa di vector database lain.

6. Praktik: Membangun Indeks dengan ChromaDB

Langkah 1 — Instalasi

TERMINAL

```
pip install chromadb sentence-transformers pypdf langchain-text-splitters
```

💾 **Unduhan cukup besar.** `sentence-transformers` membawa PyTorch, dan model embedding diunduh saat pertama dipakai (ratusan MB). Jika laptop terbatas, jalankan di Google Colab. Versi pustaka ini cepat berubah; bila ada error API, cek dokumentasi resmi tiap pustaka.

Langkah 2 — Baca PDF dan potong menjadi chunk

```
from pathlib import Path
from pypdf import PdfReader

def baca_pdf(path):
    reader = PdfReader(path)
    return "\n".join((p.extract_text() or "") for p in reader.pages)

def potong(teks, ukuran=800, tumpang=100):
    """Pemotongan ukuran tetap dengan overlap (versi paling sederhana)."""
    hasil, mulai = [], 0
    while mulai < len(teks):
        hasil.append(teks[mulai:mulai + ukuran])
        mulai += ukuran - tumpang
    return [h for h in hasil if h.strip()]
```

Alternatif yang menjaga batas paragraf dan kalimat adalah pemotong rekursif dari LangChain (pustaka kerangka kerja yang juga menyediakan loader dan pembungkus vector store):

ALTERNATIF · LangChain text splitter

```
from langchain_text_splitters import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(chunk_size=800, chunk_overlap=100)
chunks = splitter.split_text(teks)
```

Langkah 3 — Embed dan simpan ke ChromaDB

indeks.py (bagian 2: embed & store)

```
import chromadb
from chromadb.utils import embedding_functions

# model embedding multilingual (jalan lokal, tanpa API key)
emb = embedding_functions.SentenceTransformerEmbeddingFunction(
    model_name="paraphrase-multilingual-MiniLM-L12-v2")

client = chromadb.PersistentClient(path="./chroma_db")   # tersimpan di folder
koleksi = client.get_or_create_collection(
    name="pedoman", embedding_function=emb,
```

```
for pdf in Path("dokumen").glob("*.pdf"):
    chunks = potong(baca_pdf(pdf))
    koleksi.upsert(                                    # upsert: aman dijalar
        ids=[f"{pdf.stem}-{i}" for i in range(len(chunks))],
        documents=chunks,
        metadatas=[{"sumber": pdf.name, "chunk": i} for i in range(len(chunks))],
    )
    print(pdf.name, "→", len(chunks), "chunk")
```

> **Catatan versi:** cara mengatur metrik jarak (`metadata` di atas) berbeda antar versi ChromaDB. Jika muncul peringatan atau error, ikuti dokumentasi versi yang Anda pasang. Tanpa pengaturan ini, ChromaDB memakai jarak default yang berbeda dari cosine.

Langkah 4 — Query

cari.py

```
hasil = koleksi.query(query_texts=["Apa syarat cuti akademik?"], n_results=3)

for doc, meta, jarak in zip(hasil["documents"][0], hasil["metadatas"][0], hasil["dis
    print(f"[{meta['sumber']} · chunk {meta['chunk']}]  jarak={jarak:.3f}")
    print(doc[:200], "...\n")
```

Ingat: pada `distances`, **nilai lebih kecil berarti lebih mirip**. Metadata (`sumber`, `chunk`) inilah yang nanti dipakai di P-11 untuk menampilkan sumber jawaban.

7. Mengevaluasi Retrieval Sebelum Menambah LLM

Kesalahan RAG paling sering berasal dari retrieval, bukan dari LLM: jika chunk yang benar tidak ikut terambil, LLM sebaik apa pun tidak bisa menjawab dengan benar. Karena itu retrieval dievaluasi

METRIK	DEFINISI	MEMBACA HASILNYA
Hit@k	Proporsi pertanyaan uji yang chunk jawaban benarnya muncul di antara k hasil teratas	Hit@3 = 0,80 berarti 8 dari 10 pertanyaan terjawab "terambil" dalam 3 hasil teratas
MRR (Mean Reciprocal Rank)	Rata-rata dari 1/peringkat chunk benar pertama	Lebih tinggi bila chunk benar ada di peringkat atas, bukan sekadar masuk top-k

evaluasi_retrieval.py

```
# (pertanyaan, kata/frasa yang HARUS ada di chunk jawaban yang benar)
UJI = [
    ("Apa syarat cuti akademik?",          "cuti akademik"),
    ("Berapa batas kehadiran untuk ujian?", "75 persen"),
    # ... buat 15–20 pasangan dari dokumen Anda sendiri
]

def hit_at_k(uji, k=3):
    benar = 0
    for pertanyaan, kunci in uji:
        docs = koleksi.query(query_texts=[pertanyaan], n_results=k)["documents"][0]
        if any(kunci.lower() in d.lower() for d in docs):
            benar += 1
    return benar / len(uji)

for k in (1, 3, 5):
    print(f"Hit@{k} = {hit_at_k(UJI, k):.2f}")
```

💡 **Ini cara evaluasi yang praktis, bukan sempurna.** Mencocokkan kata kunci di chunk adalah pendekatan kasar: chunk bisa memuat kata kunci tetapi tidak benar-benar menjawab. Untuk laporan proyek, jelaskan keterbatasan ini dan, bila perlu, periksa sebagian hasil secara manual.

MASALAH	GEJALA	PERBAIKAN
PDF hasil scan	`extract_text()` mengembalikan teks kosong	Gunakan PDF berbasis teks, atau OCR terlebih dahulu
Model embedding tidak cocok bahasa	Hasil pencarian aneh pada teks Indonesia	Pakai model multilingual; bandingkan dengan beberapa pertanyaan uji
Chunk memotong tabel/daftar	Angka terpisah dari judul kolomnya	Pemotong rekursif atau berbasis struktur; perbesar chunk
k terlalu besar	Banyak chunk tak relevan ikut masuk	Mulai dari k = 3–5, ukur dengan Hit@k, sesuaikan
Model embedding berbeda antara indexing dan query	Hasil acak atau error dimensi	Gunakan satu `embedding_function` yang sama untuk koleksi tersebut
Tanpa metadata	Tidak bisa menyebut sumber jawaban	Simpan nama file, halaman, dan bagian sejak awal

8. Lab Terbimbing

🧪 LAB 1 — EKSPERIMEN CHUNKING (15 MENIT)

1. Baca satu PDF dengan `baca_pdf()` dan cetak 500 karakter pertama untuk memastikan teksnya terbaca baik.
2. Potong dengan `potong()` memakai tiga ukuran (mis. 500, 1000, 1500) dan overlap 10–20%. Catat jumlah chunk tiap pengaturan.
3. Lihat 3 chunk acak untuk tiap ukuran: adakah yang terpotong di tengah kalimat atau tabel?

🧪 LAB 2 — BANGUN INDEKS & CARI (25 MENIT)

2. Ajukan 5 pertanyaan dengan `query()`. Untuk tiap pertanyaan, periksa apakah chunk teratas benar-benar relevan.
3. Ajukan 2 pertanyaan dengan kata berbeda dari teks dokumen (parafrase). Apakah pencarian semantik tetap menemukannya?

💉 LAB 3 — UKUR HIT@K (25 MENIT)

1. Buat 15 pertanyaan uji beserta kata kunci jawaban dari dokumen Anda.
2. Jalankan `hit_at_k` untuk k = 1, 3, 5 pada **dua koleksi dengan ukuran chunk berbeda**.
3. Tulis kesimpulan 3–4 kalimat: ukuran mana yang lebih baik, pada pertanyaan apa gagal, dan dugaan penyebabnya.

9. Kaitan dengan Proyek Akhir

Materi hari ini adalah fondasi langsung **Proyek 4 (Chatbot Akademik berbasis RAG)**, termasuk evaluasi ukuran chunk 500, 1.000, dan 1.500 karakter. Bagi kelompok lain, konsep embedding dan pencarian semantik tetap berguna, misalnya untuk sistem rekomendasi berbasis konten (Proyek 2).

PROYEK UAS · BOBOT 35%

Panduan Proyek Akhir Kecerdasan Buatan

Lihat Proyek 4 →

Lihat deskripsi Proyek 4, format laporan, rubrik, dan timeline pengerjaan.

10. Latihan Pemahaman

No. 1 Perusahaan ingin chatbot menjawab dari SOP internal yang direvisi tiap bulan. Mengapa RAG lebih cocok daripada fine-tuning?

A Karena RAG membuat model tidak pernah salah

B Karena pembaruan cukup dengan mengganti/menambah dokumen di indeks, tanpa melatih ulang model, dan jawaban bisa merujuk sumber

C Karena RAG tidak membutuhkan dokumen sama sekali

D Karena fine-tuning tidak bisa dilakukan pada model bahasa

No. 2 Urutan yang tepat untuk alur indexing RAG adalah...

A Embed → Load → Store → Split

B Store → Split → Load → Embed

C Load → Split → Embed → Store

D Split → Store → Embed → Load

No. 3 Nilai cosine similarity antara vektor query dan sebuah chunk mendekati 1. Artinya...

A Arah kedua vektor hampir sama, sehingga maknanya sangat mirip

B Kedua teks tidak berhubungan sama sekali

D Panjang kedua teks persis sama

No. 4 Chunk berukuran sangat kecil (mis. 100 karakter) pada dokumen peraturan paling mungkin menyebabkan...

A Indeks menjadi terlalu kecil untuk disimpan

B Embedding tidak bisa dihitung

C Biaya prompt turun tanpa efek samping

D Konteks terpotong: bagian jawaban terpisah dari penjelasannya sehingga chunk yang terambil tidak cukup untuk menjawab

No. 5 Pengguna bertanya "boleh berhenti kuliah sementara?" sedangkan dokumen memakai istilah "cuti akademik". Pencarian kata kunci gagal, sedangkan pencarian dengan embedding semantik cenderung berhasil karena...

A Embedding menyalin kata persis dari pertanyaan ke dokumen

B Model embedding dilatih agar teks bermakna mirip memiliki vektor berdekatan, meski kata-katanya berbeda

C Vector database selalu menjamin hasil yang benar

D Embedding menghapus semua kata yang tidak cocok

adalah...

A Langsung mengganti LLM dengan model yang lebih besar

B Menaikkan temperature agar jawaban lebih bervariasi

C Memeriksa retrieval terpisah (mis. Hit@k): apakah chunk yang benar memang terambil sebelum sampai ke LLM?

D Menghapus metadata dari indeks

Persiapan Sesi Berikutnya

📖 MENUJU P-11: RAG PART 2 & AI UNTUK PREDIKSI/REKOMENDASI BISNIS

🔧 **Melengkapi chatbot dokumen**
Menyusun prompt dengan konteks hasil retrieval, instruksi "jawab hanya dari konteks", menampilkan sumber, dan menguji kualitas jawaban. Persiapan: simpan indeks `chroma_db` dari lab hari ini, dan pastikan Anda memiliki akses ke LLM untuk langkah generasi (API key atau opsi yang disediakan dosen).

📈 **Pengantar AI untuk bisnis**
Demand forecasting, recommendation system, dan customer segmentation. Bacaan awal: ingat kembali konsep clustering dan evaluasi regresi dari Blok 1.

📝 **Dokumentasi resmi**
Baca halaman "getting started" ChromaDB dan dokumentasi text splitter yang Anda pakai; API pustaka ini cepat berubah.

Tujuan Pembelajaran

1. Mengapa RAG?

2. Dua Alur RAG

3. Embedding & Pencarian

4. Chunking

5. Vector Database

6. Praktik ChromaDB

7. Evaluasi Retrieval

8. Lab Terbimbing

9. Kaitan Proyek Akhir

10. Latihan Pemahaman

Persiapan P-11

NAVIGASI SESI

← P-09: Prompt Engineering & API

→ P-11: RAG Part 2 & AI Bisnis

🏠 Kembali ke Beranda

🚀 Panduan Proyek Akhir

SKOR LATIHAN ANDA

0 / 6

Mulai kerjakan soal latihan!

INFO PERTEMUAN

Tipe
Praktik

Durasi
100 menit

CPMK
C3, C4

Topik
Embedding, vector database (ChromaDB), chunking, evaluasi retrieval

Prasyarat
Python 3.9+, 2–3 PDF berbasis teks

API key LLM
Tidak dibutuhkan (baru di P-11)