Lab 1 - Eksperimen Chunking

Dikerjakan dengan Python (pdfplumber + fungsi potong)

Langkah 1. Membaca PDF dan memeriksa teks

File yang dibaca: pengantar_rag.pdf. Berikut 500 karakter pertama hasil baca_pdf():

```
Pengantar Retrieval-Augmented Generation (RAG)
1. Pengertian RAG
Retrieval-Augmented Generation (RAG) adalah arsitektur yang menggabungkan pencarian
dokumen dengan model bahasa besar. Model tidak hanya mengandalkan pengetahuan hasil
pelatihan, tetapi juga menerima potongan dokumen yang relevan sebagai konteks tambahan saat
menjawab pertanyaan.
Pendekatan ini diperkenalkan oleh Lewis dan kawan-kawan pada tahun 2020 di Facebook AI
Research. Tujuannya adalah mengurangi halusinasi dan memungkinkan j
```

Pemeriksaan	Hasil
File	pengantar_rag.pdf
Total karakter	4061
Karakter rusak (U+FFFD)	0
Persentase huruf	83%

Teks terbaca bersih: tidak ada karakter rusak, kata utuh, dan urutan paragraf benar. Satu catatan: teks Tabel 1 hasil ekstraksi tercampur antar kolom karena sel yang lebih dari satu baris digabung per baris tampilan, bukan per sel.

Langkah 2. Jumlah chunk tiap pengaturan

Fungsi potong(teks, ukuran, overlap) memakai jendela geser per karakter; langkah = ukuran x (1 - overlap).

Ukuran (karakter)	overlap 10%	overlap 15%	overlap 20%
500	9	10	10
1000	5	5	5
1500	3	4	4

Tabel 1. Jumlah chunk untuk 3 ukuran x 3 overlap. Makin besar ukuran, makin sedikit chunk; makin besar overlap, makin banyak chunk.

Langkah 3. Tiga chunk acak per ukuran (overlap 15%, seed 42)

Ukuran	Jumlah chunk	Awal di tengah kalimat	Akhir di tengah kalimat	Memotong tabel
500	10	8/10 (80%)	9/10 (90%)	3/10
1000	5	3/5 (60%)	4/5 (80%)	1/5
1500	4	2/4 (50%)	3/4 (75%)	1/4

Tabel 2. Statistik seluruh chunk pada tiap ukuran (bukan hanya yang dicuplik).

Ukuran 500 karakter

Chunk #1 (karakter 0-500) | awal di tengah kalimat: tidak | akhir di tengah kalimat: ya | memotong tabel: tidak

```
Pengantar Retrieval-Augmented Generation (RAG)
1. Pengertian RAG
```

```
Retrieval-Augmented Generation (RAG) adalah arsitektur yang menggabungkan pencarian
dokumen dengan model bahasa besar. Model tidak hanya mengandalkan pengetahuan hasil
pelatihan, tetapi juga menerima potongan dokumen yang relevan sebagai konteks tambahan saat
menjawab pertanyaan.
Pendekatan ini diperkenalkan oleh Lewis dan kawan-kawan pada tahun 2020 di Facebook AI
Research. Tujuannya adalah mengurangi halusinasi dan memungkinkan j
```

Chunk #2 (karakter 425-925) | awal di tengah kalimat: ya | akhir di tengah kalimat: ya | memotong tabel: tidak

```
book AI
Research. Tujuannya adalah mengurangi halusinasi dan memungkinkan jawaban bersumber dari
dokumen terbaru tanpa harus melatih ulang model dari awal.
2. Alur Kerja RAG
Sistem RAG terdiri dari dua tahap, yaitu tahap pengindeksan yang berjalan offline dan tahap
penjawaban yang berjalan online. Pada tahap pengindeksan, dokumen dibaca, dipotong menjadi
chunk, diubah menjadi vektor embedding, lalu disimpan dalam vector store.
Pada tahap penjawaban, pertanyaan pengguna diubah menjadi vektor deng
```

Chunk #5 (karakter 1700-2200) | awal di tengah kalimat: ya | akhir di tengah kalimat: ya | memotong tabel: ya

```
tentang mobil.
4. Chunking
Chunk yang terlalu kecil kehilangan konteks, sedangkan chunk yang terlalu besar mencampur
beberapa topik sehingga skor kemiripannya menjadi encer. Praktik yang umum adalah ukuran 500
sampai 1000 karakter dengan overlap 10 sampai 20 persen.
Overlap mencegah informasi penting terpotong tepat di batas antar chunk. Pemotongan berbasis
kalimat atau paragraf lebih rapi daripada pemotongan karakter murni, tetapi ukuran chunk menjadi
tidak seragam. Tabel perlu diperlakukan khu
```

Ukuran 1000 karakter

Chunk #1 (karakter 0-1000) | awal di tengah kalimat: tidak | akhir di tengah kalimat: ya | memotong tabel: tidak

```
Pengantar Retrieval-Augmented Generation (RAG)
1. Pengertian RAG
Retrieval-Augmented Generation (RAG) adalah arsitektur yang menggabungkan pencarian
dokumen dengan model bahasa besar. Model tidak hanya mengandalkan pengetahuan hasil
pelatihan, tetapi juga menerima potongan dokumen yang relevan sebagai konteks tambahan saat
menjawab pertanyaan.
Pendekatan ini diperkenalkan oleh Lewis dan kawan-kawan pada tahun 2020 di Facebook AI
Research. Tujuannya adalah mengurangi halusinasi dan memungkinkan jawaban bersumber dari
dokumen terbaru tanpa harus melatih ulang model dari awal.
2. Alur Kerja RAG
Sistem RAG terdiri dari dua tahap, yaitu tahap pengindeksan yang berjalan offline dan tahap
penjawaban yang berjalan online. Pada tahap pengindeksan, dokumen dibaca, dipotong menjadi
chunk, diubah menjadi vektor embedding, lalu disimpan dalam vector store.
Pada tahap penjawaban, pertanyaan pengguna diubah menjadi vektor dengan model embedding
yang sama. Sistem lalu mengambil k chunk yang paling dek
```

Chunk #2 (karakter 850-1850) | awal di tengah kalimat: ya | akhir di tengah kalimat: ya | memotong tabel: tidak

```
tore.
Pada tahap penjawaban, pertanyaan pengguna diubah menjadi vektor dengan model embedding
yang sama. Sistem lalu mengambil k chunk yang paling dekat dengan vektor pertanyaan dan
menyisipkannya ke dalam prompt untuk model bahasa.
3. Embedding dan Kemiripan
Embedding adalah representasi teks berupa vektor bilangan real dengan dimensi tetap. Model
MiniLM menghasilkan embedding 384 dimensi, sedangkan model berbasis BERT-base
menghasilkan 768 dimensi. Teks dengan makna mirip akan menghasilkan vektor yang berdekatan.
Kemiripan antar vektor biasanya diukur dengan cosine similarity, yang bernilai antara -1 dan 1.
Semakin mendekati 1, semakin searah kedua vektor tersebut. Pencarian semantik tetap dapat
menemukan dokumen walaupun kata dalam pertanyaan berbeda dari teks, misalnya pertanyaan
tentang kendaraan roda empat yang cocok dengan dokumen tentang mobil.
4. Chunking
Chunk yang terlalu kecil kehilangan konteks, sedangkan chunk yang terlalu besar mencampur
beberapa topik sehingga skor kemi
```

Chunk #5 (karakter 3400-4061) | awal di tengah kalimat: ya | akhir di tengah kalimat: tidak | memotong tabel: tidak

```
i adalah MRR dan Recall@k.
Disarankan menyiapkan minimal 15 sampai 30 pertanyaan uji lengkap dengan kata kunci jawaban.
Bandingkan beberapa pengaturan chunk pada pertanyaan yang sama agar perbedaannya adil.
```

```
7. Halusinasi dan Sitasi
Halusinasi adalah keluaran model yang terdengar meyakinkan tetapi tidak didukung fakta. RAG
mengurangi halusinasi, tetapi tidak menghilangkannya sepenuhnya, terutama bila chunk yang
diambil tidak relevan.
Sistem yang baik menampilkan sumber berupa nama file dan nomor halaman sehingga pengguna
dapat memeriksa jawaban. Bila chunk teratas tidak relevan, model sebaiknya menjawab bahwa
informasi tidak ditemukan daripada mengarang.
```

Ukuran 1500 karakter

Chunk #1 (karakter 0-1500) | awal di tengah kalimat: tidak | akhir di tengah kalimat: ya | memotong tabel: tidak

```
Pengantar Retrieval-Augmented Generation (RAG)
1. Pengertian RAG
Retrieval-Augmented Generation (RAG) adalah arsitektur yang menggabungkan pencarian
dokumen dengan model bahasa besar. Model tidak hanya mengandalkan pengetahuan hasil
pelatihan, tetapi juga menerima potongan dokumen yang relevan sebagai konteks tambahan saat
menjawab pertanyaan.
Pendekatan ini diperkenalkan oleh Lewis dan kawan-kawan pada tahun 2020 di Facebook AI
Research. Tujuannya adalah mengurangi halusinasi dan memungkinkan jawaban bersumber dari
dokumen terbaru tanpa harus melatih ulang model dari awal.
2. Alur Kerja RAG
Sistem RAG terdiri dari dua tahap, yaitu tahap pengindeksan yang berjalan offline dan tahap
penjawaban yang berjalan online. Pada tahap pengindeksan, dokumen dibaca, dipotong menjadi
chunk, diubah menjadi vektor embedding, lalu disimpan dalam vector store.
Pada tahap penjawaban, pertanyaan pengguna diubah menjadi vektor dengan model embedding
yang sama. Sistem lalu mengambil k chunk yang paling dekat dengan vektor pertanyaan dan
menyisipkannya ke dalam prompt untuk model bahasa.
3. Embedding dan Kemiripan
Embedding adalah representasi teks berupa vektor bilangan real dengan dimensi tetap. Model
MiniLM menghasilkan embedding 384 dimensi, sedangkan model berbasis BERT-base
menghasilkan 768 dimensi. Teks dengan makna mirip akan menghasilkan vektor yang berdekatan.
Kemiripan antar vektor biasanya diukur dengan cosine similarity, yang bernilai antara -1 dan 1.
Semakin mendekati 1, semakin sear
```

Chunk #3 (karakter 2550-4050) | awal di tengah kalimat: tidak | akhir di tengah kalimat: ya | memotong tabel: ya

```
dan presisi topik
1500 karakter Konteks luas Skor encer dan boros Dokumen naratif panjang
token
Sumber: ringkasan penulis.
5. Vector Store
Vector store menyimpan embedding dan mendukung pencarian tetangga terdekat. FAISS adalah
pustaka buatan Meta untuk pencarian tetangga terdekat yang sangat cepat. ChromaDB menyimpan
embedding bersama metadata seperti nama file dan nomor halaman, sedangkan Qdrant
menyediakan layanan server tersendiri.
Untuk koleksi besar, banyak vector store memakai indeks HNSW yang mencari tetangga terdekat
secara aproksimasi demi kecepatan, dengan sedikit pengorbanan akurasi.
6. Evaluasi Retrieval
Hit@k adalah proporsi pertanyaan uji yang jawaban benarnya muncul di antara k chunk teratas.
Sebagai contoh, bila 12 dari 15 pertanyaan terjawab pada lima chunk teratas, maka Hit@5 bernilai
0,8. Metrik lain yang sering dipakai adalah MRR dan Recall@k.
Disarankan menyiapkan minimal 15 sampai 30 pertanyaan uji lengkap dengan kata kunci jawaban.
Bandingkan beberapa pengaturan chunk pada pertanyaan yang sama agar perbedaannya adil.
7. Halusinasi dan Sitasi
Halusinasi adalah keluaran model yang terdengar meyakinkan tetapi tidak didukung fakta. RAG
mengurangi halusinasi, tetapi tidak menghilangkannya sepenuhnya, terutama bila chunk yang
diambil tidak relevan.
Sistem yang baik menampilkan sumber berupa nama file dan nomor halaman sehingga pengguna
dapat memeriksa jawaban. Bila chunk teratas tidak relevan, model sebaiknya menjawab bahwa
informasi tidak ditemukan daripada
```

Chunk #4 (karakter 3825-4061) | awal di tengah kalimat: ya | akhir di tengah kalimat: tidak | memotong tabel: tidak

```
ak relevan.
Sistem yang baik menampilkan sumber berupa nama file dan nomor halaman sehingga pengguna
dapat memeriksa jawaban. Bila chunk teratas tidak relevan, model sebaiknya menjawab bahwa
informasi tidak ditemukan daripada mengarang.
```

Kesimpulan

Pemotongan per karakter tidak mengenal batas kalimat maupun tabel. Hasil seluruh chunk (overlap 15%) - ukuran 500: awal 8/10 (80%), akhir 9/10 (90%), memotong tabel 3/10; ukuran 1000: awal 3/5 (60%), akhir 4/5 (80%), memotong tabel 1/5; ukuran 1500: awal 2/4 (50%), akhir 3/4 (75%), memotong tabel 1/4. Jadi pada setiap ukuran, mayoritas chunk berakhir di tengah kalimat dan sebagian besar juga dimulai di tengah kalimat (hanya chunk pertama yang selalu bersih). Persentasenya tidak turun banyak saat ukuran diperbesar; yang berubah adalah jumlah chunk dan banyaknya topik per chunk. Tabel 1 terpotong oleh sedikitnya satu chunk pada tiap ukuran, dan teks tabel hasil ekstraksi sudah bercampur antar kolom, sehingga baris tabel yang terpisah dari baris judulnya sulit dipahami. Perbaikan yang layak dicoba: pemotongan berbasis kalimat/paragraf dan memperlakukan tabel sebagai satu chunk utuh.

Lampiran: kode lab1_chunking.py

```
"""Lab 1 - Eksperimen chunking (15 menit).

1. Baca satu PDF dengan baca_pdf() dan cetak 500 karakter pertama.
2. Potong dengan potong() memakai tiga ukuran (500, 1000, 1500) dan overlap 10-20%.
3. Lihat 3 chunk acak tiap ukuran: terpotong di tengah kalimat atau tabel?
"""
import os
import random
import sys

sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
from laporan_utils import Laporan
from rag_utils import FOLDER_DOKUMEN, baca_pdf, norm, potong, potong_dengan_posisi

BASE = os.path.dirname(os.path.abspath(__file__))
PDF = os.path.join(FOLDER_DOKUMEN, "pengantar_rag.pdf")   # ganti dengan PDF Anda
UKURAN = [500, 1000, 1500]
OVERLAP = [0.10, 0.15, 0.20]
OVERLAP_UTAMA = 0.15
SEED = 42


def wilayah_tabel(teks):
    """Cari rentang karakter tabel ("Tabel N." sampai baris "Sumber:")."""
    hasil, pos = [], 0
    while True:
        a = teks.find("Tabel ", pos)
        if a < 0:
            return hasil
        b = teks.find("Sumber:", a)
        b = len(teks) if b < 0 else teks.find("\n", b)
        hasil.append((a, b))
        pos = b


def awal_di_tengah_kalimat(teks, mulai):
    """Chunk dimulai di tengah kalimat bila karakter bermakna sebelumnya bukan akhir kalimat/baris."""
    sebelum = teks[:mulai].rstrip(" ")
    return bool(sebelum) and sebelum[-1] not in ".!?:\n"


def akhir_di_tengah_kalimat(teks, akhir):
    ujung = teks[:akhir].rstrip()
    return bool(ujung) and ujung[-1] not in ".!?"


def memotong_tabel(wil, mulai, akhir):
    """True bila chunk beririsan dengan tabel tetapi tidak memuat seluruh tabel."""
    for a, b in wil:
        beririsan = mulai < b and akhir > a
        utuh = mulai <= a and akhir >= b
        if beririsan and not utuh:
            return True
    return False


def main():
    # ---- Langkah 1
    teks = baca_pdf(PDF)
    n_huruf = sum(c.isalpha() for c in teks)
    kualitas = {
        "File": os.path.basename(PDF),
        "Total karakter": len(teks),
        "Karakter rusak (U+FFFD)": teks.count("\ufffd"),
        "Persentase huruf": f"{n_huruf / len(teks):.0%}",
    }
    print("=== Langkah 1: 500 karakter pertama ===")
    print(teks[:500])
    print(kualitas)

    # ---- Langkah 2
    print("\n=== Langkah 2: jumlah chunk ===")
    tabel_jumlah = [["Ukuran (karakter)"] + [f"overlap {int(o * 100)}%" for o in OVERLAP]]
    for u in UKURAN:
        baris = [u]
        for o in OVERLAP:
            n = len(potong(teks, u, o))
            baris.append(n)
            print(f"ukuran={u:5d} overlap={o:.0%} -> {n} chunk")
        tabel_jumlah.append(baris)
```

```
    # ---- Langkah 3
    wil = wilayah_tabel(teks)
    rnd = random.Random(SEED)
    cuplikan, statistik = {}, [["Ukuran", "Jumlah chunk", "Awal di tengah kalimat",
                                "Akhir di tengah kalimat", "Memotong tabel"]]
    for u in UKURAN:
        pot = potong_dengan_posisi(teks, u, OVERLAP_UTAMA)
        n = len(pot)
        a = sum(awal_di_tengah_kalimat(teks, m) for m, _, _ in pot)
        z = sum(akhir_di_tengah_kalimat(teks, e) for _, e, _ in pot)
        t = sum(memotong_tabel(wil, m, e) for m, e, _ in pot)
        statistik.append([u, n, f"{a}/{n} ({a / n:.0%})", f"{z}/{n} ({z / n:.0%})", f"{t}/{n}"])
        pilih = sorted(rnd.sample(range(n), min(3, n)))
        cuplikan[u] = [(i + 1, pot[i][0], pot[i][1], pot[i][2],
                        awal_di_tengah_kalimat(teks, pot[i][0]),
                        akhir_di_tengah_kalimat(teks, pot[i][1]),
                        memotong_tabel(wil, pot[i][0], pot[i][1])) for i in pilih]
        print(f"\n=== Langkah 3: ukuran {u} (3 chunk acak) ===")
        for no, m, e, isi, aw, ak, tb in cuplikan[u]:
            print(f"--- chunk #{no} [{m}:{e}] awal_tengah={aw} akhir_tengah={ak} potong_tabel={tb}")
            print(isi)

    # ---- Kesimpulan otomatis dari angka
    rincian = "; ".join(
        f"ukuran {r[0]}: awal {r[2]}, akhir {r[3]}, memotong tabel {r[4]}" for r in statistik[1:])
    kesimpulan = (
        "Pemotongan per karakter tidak mengenal batas kalimat maupun tabel. Hasil seluruh chunk (overlap
    "
        f"{int(OVERLAP_UTAMA * 100)}%) - {rincian}. Jadi pada setiap ukuran, mayoritas chunk berakhir di
    tengah "
        "kalimat dan sebagian besar juga dimulai di tengah kalimat (hanya chunk pertama yang selalu
    bersih). "
        "Persentasenya tidak turun banyak saat ukuran diperbesar; yang berubah adalah jumlah chunk dan "
        "banyaknya topik per chunk. Tabel 1 terpotong oleh sedikitnya satu chunk pada tiap ukuran, dan
    teks "
        "tabel hasil ekstraksi sudah bercampur antar kolom, sehingga baris tabel yang terpisah dari baris
     "
        "judulnya sulit dipahami. Perbaikan yang layak dicoba: pemotongan berbasis kalimat/paragraf dan "
        "memperlakukan tabel sebagai satu chunk utuh."
    )
    print("\nKesimpulan:", kesimpulan)

    # ---- Laporan PDF
    L = Laporan(os.path.join(BASE, "Lab1_Eksperimen_Chunking.pdf"),
                "Lab 1 - Eksperimen Chunking", "Dikerjakan dengan Python (pdfplumber + fungsi potong)")
    L.h1("Langkah 1. Membaca PDF dan memeriksa teks")
    L.p(f"File yang dibaca: {os.path.basename(PDF)}. Berikut 500 karakter pertama hasil baca_pdf():")
    L.kode(teks[:500])
    L.spasi()
    L.tabel([["Pemeriksaan", "Hasil"]] + [[k, v] for k, v in kualitas.items()], [6 * 28.35, 8 * 28.35])
    L.p("Teks terbaca bersih: tidak ada karakter rusak, kata utuh, dan urutan paragraf benar. "
        "Satu catatan: teks Tabel 1 hasil ekstraksi tercampur antar kolom karena sel yang lebih dari satu
     baris "
        "digabung per baris tampilan, bukan per sel.")
    L.h1("Langkah 2. Jumlah chunk tiap pengaturan")
    L.p("Fungsi potong(teks, ukuran, overlap) memakai jendela geser per karakter; langkah = ukuran x (1 -
     overlap).")
    L.tabel(tabel_jumlah)
    L.cap("Tabel 1. Jumlah chunk untuk 3 ukuran x 3 overlap. Makin besar ukuran, makin sedikit chunk; "
          "makin besar overlap, makin banyak chunk.")
    L.h1(f"Langkah 3. Tiga chunk acak per ukuran (overlap {int(OVERLAP_UTAMA * 100)}%, seed {SEED})")
    L.tabel(statistik)
    L.cap("Tabel 2. Statistik seluruh chunk pada tiap ukuran (bukan hanya yang dicuplik).")
    for u in UKURAN:
        L.h2(f"Ukuran {u} karakter")
        for no, m, e, isi, aw, ak, tb in cuplikan[u]:
            L.p(f"Chunk #{no} (karakter {m}-{e}) | awal di tengah kalimat: {'ya' if aw else 'tidak'} | "
                f"akhir di tengah kalimat: {'ya' if ak else 'tidak'} | memotong tabel: {'ya' if tb else
    'tidak'}")
            L.kode(isi)
            L.spasi(4)
    L.h1("Kesimpulan")
    L.p(kesimpulan)
    L.halaman_baru()
    L.h1("Lampiran: kode lab1_chunking.py")
    L.kode(open(__file__, encoding="utf-8").read())
    L.simpan()
    print("\nLaporan: Lab1_Eksperimen_Chunking.pdf")


if __name__ == "__main__":
    main()
```