Meeting 12: RAG — Retrieval-Augmented Generation

AI-40X: Generative AI & Large Language Models

Hendri Karisma, M.T.
Dosen Teknik Informatika STMIK Tazkia
VP Engineering at jejakin.com, 2026

Semester 4 — 2025/2026

Outline

Knowledge Cutoff: LLM Tidak Tahu Segalanya

- LLM dilatih pada data hingga tanggal tertentu (**knowledge cutoff**).
  - GPT-4o: data hingga Oktober 2023
  - Llama 3: data hingga Maret 2023
- **Pertanyaan setelah cutoff:** "Siapa yang memenangkan Pilpres 2024?" $\rightarrow$ LLM tidak tahu.
- **Data privat:** "Berapa total penjualan PT ABC bulan lalu?" $\rightarrow$ LLM tidak pernah melihat data ini.
- Fine-tuning **bukan solusi yang tepat** untuk masalah ini:
  - Mahal dan lambat untuk setiap update data
  - LLM cenderung menghafal, bukan "membaca" saat inference
  - Tidak cocok untuk fakta yang sering berubah

Halusinasi: Percaya Diri tapi Salah

- **Halusinasi** = LLM menghasilkan jawaban yang terdengar meyakinkan, tapi faktanya **salah**.
- Contoh nyata:
  - Membuat kutipan jurnal yang tidak pernah ada
  - Menyebutkan pasal hukum yang tidak ada dalam UU
  - Memberikan statistik yang salah dengan percaya diri
- **Mengapa terjadi?** LLM adalah model probabilistik — ia memprediksi token berikutnya yang *paling mungkin*, bukan yang paling *benar*.
- Halusinasi adalah masalah fundamental, bukan bug yang bisa diperbaiki dengan prompt saja.

Analogi RAG: Ujian Buku Terbuka vs Buku Tertutup

Tanpa RAG (Closed-Book Exam)

- LLM menjawab hanya dari "ingatan" (parameter model).
- Jika tidak ingat → mengarang.
- Tidak bisa diverifikasi sumbernya.

Dengan RAG (Open-Book Exam)

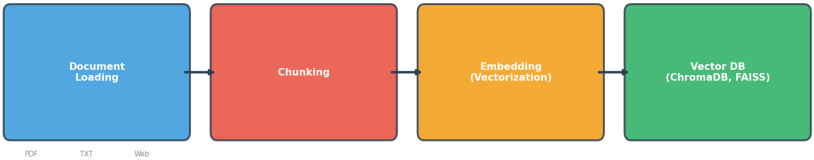
- LLM diberi "buku referensi" (dokumen relevan) sebelum menjawab.
- Menjawab berdasarkan teks yang diberikan.
- Bisa dicek sumbernya (traceability).

Definisi RAG

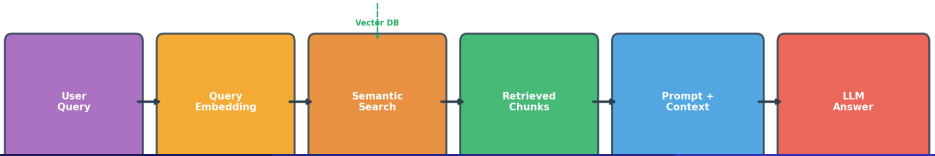
Retrieval-Augmented Generation = Ambil dokumen relevan (*Retrieval*), masukkan ke prompt (*Augmented*), baru LLM menjawab (*Generation*).

RAG Pipeline: 6 Langkah

Fase 1: Indexing (Offline)



Fase 2: Query (Online)



Fase Offline: Indexing (Langkah 1–3)

- Dilakukan **sekali** saat data baru masuk (bukan setiap query).

Pseudocode: Indexing Pipeline

```
# Langkah 1: Load dokumen
from langchain_community.document_loaders import PyMuPDFLoader
docs = PyMuPDFLoader("laporan_keuangan.pdf").load()

# Langkah 2: Chunking
from langchain.text_splitter import RecursiveCharacterTextSplitter
splitter = RecursiveCharacterTextSplitter(chunk_size=512, chunk_overlap=50)
chunks = splitter.split_documents(docs)

# Langkah 3: Embedding + simpan ke Vector DB
from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import HuggingFaceEmbeddings

embedding_model = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
vectorstore = Chroma.from_documents(chunks, embedding_model, persist_directory="./db")
```

Fase Online: Retrieval & Generation (Langkah 4–6)

- Dilakukan **setiap kali** user mengirim pertanyaan.

Pseudocode: Query Pipeline

```
# Langkah 4: Retrieval - cari chunk relevan
query = "Berapa total revenue Q3 2024?"
retriever = vectorstore.as_retriever(search_kwargs={"k": 5})
relevant_chunks = retriever.invoke(query)

# Langkah 5: Augmentation - gabungkan ke prompt
context = "\n".join([chunk.page_content for chunk in relevant_chunks])
prompt = f"""Berdasarkan konteks berikut, jawab pertanyaan user.
Jika jawabannya tidak ada di konteks, katakan "Saya tidak menemukan informasinya."

Konteks:
{context}

Pertanyaan: {query}
Jawaban:"""

# Langkah 6: Generation
response = llm.invoke(prompt)
```

Mengapa Chunking Penting?

- Dokumen PDF bisa ratusan halaman — terlalu besar untuk context window LLM.
- Kita perlu memecah (**chunk**) dokumen menjadi potongan yang:
  1. Cukup kecil untuk masuk context window
  2. Cukup besar untuk tetap bermakna
  3. Memiliki *overlap* agar tidak kehilangan konteks di perbatasan
- **Chunking adalah seni, bukan sains.** Parameter harus dieksperimen.

Strategi Chunking

1. Fixed-Size Chunking

- Potong setiap N token (misal 512).
- Tambahkan *overlap* (misal 50 token).
- Sederhana, tapi bisa memotong di tengah kalimat.

2. Recursive Text Splitting

- Coba potong per paragraf dulu.
- Jika masih terlalu besar, potong per kalimat.
- Jika masih besar, potong per kata.
- **Paling umum digunakan** (default LangChain).

3. Semantic Chunking

- Hitung embedding tiap kalimat.
- Potong jika similarity antar kalimat berturutan turun drastis.
- Lebih cerdas, tapi lebih lambat.

Trade-off Chunk Size:

- Terlalu kecil (100 token): kehilangan konteks, banyak chunk tidak bermakna.
- Terlalu besar (2000 token): noise, embedding kurang spesifik.
- **Sweet spot**: 256–1024 token.

Template Prompt RAG: Context Injection

System Prompt untuk RAG

```
SYSTEM_PROMPT = """Anda adalah asisten AI yang membantu menjawab pertanyaan
berdasarkan dokumen yang diberikan.

ATURAN PENTING:
1. Jawab HANYA berdasarkan konteks yang diberikan di bawah.
2. Jika jawaban tidak ditemukan dalam konteks, katakan:
   "Maaf, saya tidak menemukan informasi tersebut dalam dokumen yang tersedia."
3. Jangan mengarang atau menambahkan informasi di luar konteks.
4. Sebutkan bagian dokumen mana yang menjadi sumber jawaban Anda.
"""
```

User Prompt Template

```
USER_PROMPT = """Konteks dari dokumen:
---
{retrieved_chunks}
---

Pertanyaan: {user_question}

Jawaban (berdasarkan konteks di atas):"""
```

Menangani "Saya Tidak Tahu"

- Salah satu tantangan terbesar RAG: LLM tetap bisa berhalusinasi meski diberi konteks.
- **Strategi mencegah halusinasi:**
  1. **Instruksi eksplisit** di system prompt: "Jika tidak ada di konteks, bilang tidak tahu."
  2. **Confidence check:** Minta LLM memberikan skor keyakinan (1–5).
  3. **Citation:** Minta LLM menyertakan kutipan langsung dari konteks.
  4. **Guardrail post-processing:** Cek apakah jawaban berisi informasi yang tidak ada di chunk.
- **Prinsip:** Lebih baik LLM bilang "tidak tahu" daripada berhalusinasi.

Re-Ranking dengan Cross-Encoder

- **Masalah:** Retrieval awal (bi-encoder) cepat tapi kurang presisi.
- **Solusi: Two-stage retrieval**
  1. **Stage 1 (Recall):** Ambil top-50 chunk dengan bi-encoder (cepat).
  2. **Stage 2 (Precision):** Re-rank 50 chunk tersebut dengan cross-encoder (akurat), ambil top-5.
- **Bi-Encoder:** Encode query dan dokumen secara terpisah $\rightarrow$ bandingkan vektor. Cepat, tapi interaksi terbatas.
- **Cross-Encoder:** Encode query + dokumen bersamaan $\rightarrow$ skor relevansi langsung. Lambat, tapi sangat akurat.
- Model: `cross-encoder/ms-marco-MiniLM-L-6-v2` (gratis, CPU-friendly).

HyDE: Hypothetical Document Embeddings

- **Masalah:** Pertanyaan user seringkali pendek dan berbeda gaya dari dokumen.
  - Query: "apa syarat NPWP?"
  - Dokumen: "Pendaftaran Nomor Pokok Wajib Pajak memerlukan dokumen berupa..."
- **Ide HyDE (Query Expansion):**
  1. Minta LLM *membayangkan* jawaban hipotetis (tanpa sumber).
  2. Gunakan jawaban hipotetis itu sebagai query pencarian.
  3. Jawaban hipotetis lebih mirip gaya penulisan dokumen → retrieval lebih baik.
- **Kelemahan:** Menambah 1 kali panggilan LLM (latency naik). Tidak selalu lebih baik.

Lost-in-the-Middle Problem

- **Penemuan (Liu et al., 2023):** LLM cenderung lebih memperhatikan informasi di **awal** dan **akhir** prompt, dan melupakan informasi di **tengah**.
- Ini berbahaya jika chunk paling relevan berada di posisi tengah dari context!
- **Solusi praktis:**
  1. Kurangi jumlah chunk yang dimasukkan (5 lebih baik dari 20).
  2. Urutkan chunk terpenting di **awal** dan **akhir** prompt.
  3. Gunakan re-ranking untuk memastikan hanya chunk berkualitas yang masuk.

Multi-hop RAG: Pertanyaan Kompleks

- **Pertanyaan sederhana:** "Berapa gaji CEO?" $\rightarrow$ 1 kali retrieval cukup.
- **Pertanyaan multi-hop:** "Apakah total pengeluaran divisi IT lebih besar dari revenue Q3?"
  - Butuh retrieval #1: total pengeluaran divisi IT
  - Butuh retrieval #2: revenue Q3
  - Baru bisa membandingkan keduanya
- **Strategi:**
  1. **Query Decomposition:** Pecah pertanyaan kompleks menjadi sub-pertanyaan.
  2. **Iterative Retrieval:** Lakukan retrieval bertahap, gunakan jawaban sebelumnya untuk query berikutnya.
  3. **Agentic RAG:** LLM memutuskan sendiri kapan perlu retrieval tambahan (menggunakan tools/function calling).

Lab: Membangun RAG Pipeline

Tools yang digunakan:

- **LangChain** — framework orchestrasi RAG
- **ChromaDB** — vector database (dari Meeting 11)
- **Sentence-BERT** — model embedding (gratis, lokal)
- **Groq API / Ollama** — LLM (gratis!)

Setup Environment

```
pip install langchain langchain-community chromadb sentence-transformers pymupdf
pip install langchain-groq   # Jika pakai Groq (gratis, cepat, cloud)
# ATAU
# Install Ollama dari ollama.com, lalu: ollama pull llama3.2:1b
```

Tidak Perlu GPU atau API Berbayar!

Groq menyediakan API gratis dengan rate limit. Ollama menjalankan model kecil (1B–3B) langsung di laptop. Pilih salah satu.

Lab: Arsitektur yang Akan Dibangun

1. **Input:** Dokumen PDF (misal: RPS mata kuliah, buku, laporan)
2. **Proses Indexing:**
   - Load PDF → Chunk (RecursiveTextSplitter, 512 token, overlap 50)
   - Embed dengan `all-MiniLM-L6-v2` → Simpan ke ChromaDB
3. **Proses Query:**
   - User bertanya → Retrieve top-5 chunks → Inject ke prompt
   - LLM (Groq/Ollama) menjawab berdasarkan konteks
4. **Output:** Jawaban + sumber (halaman/chunk mana yang digunakan)

Tantangan Bonus

Tambahkan fitur: (1) Hybrid search (BM25 + vector), (2) Re-ranking dengan cross-encoder, (3) Streamlit UI sederhana.

Kesimpulan

1. **Halusinasi** adalah masalah fundamental LLM — RAG adalah solusi paling praktis saat ini.
2. **RAG Pipeline** terdiri dari 6 langkah: Load $\rightarrow$ Chunk $\rightarrow$ Embed $\rightarrow$ Retrieve $\rightarrow$ Augment $\rightarrow$ Generate.
3. **Chunking** sangat mempengaruhi kualitas RAG — eksperimen dengan ukuran dan strategi.
4. **Prompt design** yang baik mencegah LLM berhalusinasi meski sudah diberi konteks.
5. **Advanced techniques** (re-ranking, HyDE, multi-hop) meningkatkan kualitas secara signifikan.
6. RAG $\neq$ sempurna — tetap butuh evaluasi dan iterasi.

Referensi

- Lewis, P., et al. (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." *NeurIPS 2020*.
- Liu, N., et al. (2023). "Lost in the Middle: How Language Models Use Long Contexts." *arXiv:2307.03172*.
- Gao, L., et al. (2022). "Precise Zero-Shot Dense Retrieval without Relevance Labels" (HyDE). *arXiv:2212.10496*.
- LangChain Documentation: `https://python.langchain.com`
- ChromaDB Documentation: `https://docs.trychroma.com`
- Groq API: `https://console.groq.com` (gratis)