

RAG

Apa itu RAG?

RAG (Retrieval-Augmented Generation) adalah teknik di mana *Large Language Model (LLM)* seperti Ollama tidak hanya menjawab berdasarkan apa yang sudah dilatih sebelumnya, tapi **diperkuat dengan informasi dari luar**, seperti file PDF, dokumen teks, database, dll.

Dengan RAG:

- Kamu upload dokumen-dokumenmu.
- Sistem akan mengambil potongan informasi yang relevan dari dokumen tsb (retrieval).
- Lalu menggabungkan potongan itu ke dalam prompt ke LLM untuk dijawab (augmented generation).

Cara kerja RAG (secara teknis)

1. **Ingest dokumen**
File seperti PDF, DOCX, TXT, dst dibaca dan diubah jadi potongan teks kecil (chunks).
2. **Buat embedding**
Setiap chunk diubah menjadi *vector embedding* pakai model seperti all-MiniLM (atau model embedding lokal kalau mau offline total).
3. **Simpan embedding di vector database**
Contoh: FAISS, Chroma, Weaviate, Pinecone, dll. Kalau offline: pakai FAISS atau Chroma.
4. **Saat ada pertanyaan user**
Pertanyaan juga diubah jadi embedding, lalu dibandingkan dengan yang ada di database untuk nyari potongan dokumen paling relevan.
5. **Gabungkan hasil**
Potongan dokumen itu dikirimkan bareng pertanyaan ke LLM (dalam hal ini: Ollama) untuk dijawab.

Alur data RAG sederhana

[User Input]



[Convert ke embedding]



[Search di vector DB → dapat potongan relevan]



[Gabungkan ke prompt → kirim ke Ollama]



[Ollama jawab pakai info + konteks dokumen]

Stack minimal untuk lokal RAG (offline friendly)

Komponen	Tool/Library
LLM	Ollama (ex: llama3, mistral, dll)
Embedding model sentence-transformers atau ollama embedding	
Vector DB	Chroma atau FAISS
Backend API	Python Flask/FastAPI
Frontend	Sudah kamu punya (PHP/HTML/JS)

Mau lanjut ke mana?

Kalau kamu mau lanjut, saya bisa bantu:

1. **Buat contoh RAG sederhana** pakai Python + Ollama + Chroma.
2. **Integrasi ke frontend kamu sekarang.**
3. **Upload dokumen dan mulai tanya jawab berdasarkan dokumen.**

Kamu pilih mau mulai dari mana? Atau aku bikinkan semua bertahap satu per satu?

Membuat RAG Sederhana

Siap, kita buat **RAG sederhana** dulu — full lokal, bisa kamu jalankan di notebook kamu. Kita pakai:

- **Ollama** (untuk menjawab)
- **Langchain** (untuk workflow)
- **Chroma** (untuk vector DB)
- **PyMuPDF / pdfplumber / llama-index** (untuk baca PDF)
- **Python + Flask/FastAPI** (opsional untuk backend, nanti)

Persiapan

Pastikan kamu sudah install Python. Sekarang install dulu package yang diperlukan:

`pip install langchain chromadb sentence-transformers unstructured pdfplumber ollama`

Kalau kamu punya file PDF atau TXT buat uji coba, siapin dulu di folder.

Struktur File Minimal

rag_demo/

├─ app.py <-- script utama

├─ documents/

| └─ contoh.pdf <-- file yang akan di-query

app.py – Prototipe RAG sederhana

```
from langchain.document_loaders import PyPDFLoader
```

```
from langchain.text_splitter import CharacterTextSplitter
```

```
from langchain.vectorstores import Chroma
```

```
from langchain.embeddings import HuggingFaceEmbeddings
```

```
from langchain.llms import Ollama
```

```
from langchain.chains import RetrievalQA
```

1. Load PDF

```
loader = PyPDFLoader("documents/contoh.pdf")
```

```
documents = loader.load()
```

2. Split jadi potongan

```
text_splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=100)
```

```
texts = text_splitter.split_documents(documents)
```

3. Buat Embedding & simpan di Chroma

```
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
```

```
db = Chroma.from_documents(texts, embedding_model, persist_directory="db")
```

```
db.persist()
```

4. Setup LLM (pakai Ollama lokal, contoh: llama3)

```
llm = Ollama(model="llama3")
```

5. Buat sistem tanya jawab

```
qa = RetrievalQA.from_chain_type(llm=llm, retriever=db.as_retriever())
```

6. Jalankan sesi tanya jawab

```
while True:
```

```
    query = input("\nPertanyaan kamu: ")
```

```
    if query.lower() in ['exit', 'quit']:
```

```
        break
```

```
    answer = qa.run(query)
```

```
    print("\nJawaban:\n", answer)
```

Jalankan:

python app.py

Lalu kamu bisa tanya:

Pertanyaan kamu: Apa isi utama dokumen ini?

Ollama akan menjawab berdasarkan isi dokumen PDF yang kamu masukkan.

Kalau sudah berhasil, next step kita bisa:

- Tambah upload file baru
- Pakai antarmuka web (Flask / PHP-manggil-API)
- Multi-dokumen support
- Riwayat percakapan

Kalau kamu mau aku bantuin buat versi Flask + Web frontend-nya sekalian, tinggal bilang.
Mau lanjut? 😊

Lanjut versi Flask & Web Front End

Mantap! Sekarang kita lanjut ke **versi Web (Flask)** agar kamu bisa akses dari browser dan mungkin nanti diintegrasikan ke frontend PHP atau HTML kamu.

Tujuan

Bikin antarmuka web sederhana di mana user:

1. **Upload dokumen (PDF)**
 2. **Bisa tanya jawab** berdasarkan isi dokumen tsb
 3. **Jawaban dihasilkan oleh Ollama + RAG**
-

Struktur Project

rag_web/

```
└─ app.py
└─ templates/
|   └─ index.html
└─ documents/
|   └─ (tempat file PDF di-upload)
└─ db/          <-- Chroma vector DB
```

app.py

```
from flask import Flask, request, render_template
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import Chroma
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.llms import Ollama
from langchain.chains import RetrievalQA
import os

app = Flask(__name__)

UPLOAD_FOLDER = "documents"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# Setup LLM dan Embedding
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-
MiniLM-L6-v2")
llm = Ollama(model="llama3")

def load_db():
    return Chroma(persist_directory="db", embedding_function=embedding_model)
```

```

@app.route("/", methods=["GET", "POST"])
def index():
    answer = ""

    if request.method == "POST":
        if 'file' in request.files:
            file = request.files['file']

            filepath = os.path.join(UPLOAD_FOLDER, file.filename)

            file.save(filepath)

            loader = PyPDFLoader(filepath)

            documents = loader.load()

            splitter = CharacterTextSplitter(chunk_size=500, chunk_overlap=100)
            texts = splitter.split_documents(documents)

            db = Chroma.from_documents(texts, embedding_model, persist_directory="db")
            db.persist()

        elif 'question' in request.form:
            question = request.form['question']

            db = load_db()

            qa = RetrievalQA.from_chain_type(llm=llm, retriever=db.as_retriever())

            answer = qa.run(question)

    return render_template("index.html", answer=answer)

if __name__ == "__main__":

```

```
app.run(debug=True)
```

templates/index.html

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>RAG Chatbot</title>
```

```
</head>
```

```
<body>
```

```
  <h1>RAG Chatbot dengan Ollama</h1>
```

```
  <h2>Upload Dokumen (PDF)</h2>
```

```
  <form action="/" method="post" enctype="multipart/form-data">
```

```
    <input type="file" name="file" />
```

```
    <input type="submit" value="Upload" />
```

```
  </form>
```

```
  <h2>Tanya Dokumen</h2>
```

```
  <form action="/" method="post">
```

```
    <input type="text" name="question" style="width: 400px;" placeholder="Tulis pertanyaan di sini" />
```

```
    <input type="submit" value="Tanya" />
```

```
  </form>
```

```
{% if answer %}
```

```
  <h3>Jawaban:</h3>
```

```
  <p>{{ answer }}</p>
```

```
{% endif %}
```


</body>

</html>

Jalankan Aplikasi

`python app.py`

Buka browser ke `http://127.0.0.1:5000`

Next?

Mau lanjut ke:

- UI lebih cantik (pakai Bootstrap atau Tailwind)?
- Upload multi file?
- Simpan riwayat Q&A?
- Integrasi dengan frontend PHP kamu?

Tinggal bilang, bisa kita lanjutkan step by step.