🖋️ Kafka Conversational AI (RAG-Based System)

A Retrieval-Augmented Generation (RAG) conversational system that simulates Franz Kafka's voice ar
worldview using his literary works, letters, and biographical materials.

Built with **LangChain + ChromaDB + Ollama + Streamlit**.

📌 Project Overview

This project implements a conversational AI system that:

- Retrieves relevant passages from Kafka's works
- Grounds responses in real textual sources
- Simulates Kafka's tone and existential style
- Displays source transparency in the UI

The system is designed for:

- Academic experimentation
- RAG architecture practice
- Persona-based LLM systems
- Transparent source-backed AI conversations

🏗️ Architecture

User Input
→ Streamlit UI
→ `kafka_rag_answer()`
→ ChromaDB Retriever
→ Ollama LLM
→ Grounded Response + Sources
→ UI Display (Answer + Source Panel)

Core Components

1️⃣ Ingestion Pipeline

- Load Kafka texts (PDF / TXT)
- Split into chunks
- Add metadata (author, work, type, chunk_id)

- Generate embeddings
- Store in ChromaDB

2️⃣ Retrieval Layer

- Semantic similarity search
- Optional metadata filtering
- Returns top-k relevant chunks

3️⃣ Generation Layer

- Prompt enforces grounding
- Uses retrieved context only
- Produces stylistically aligned output

4️⃣ Streamlit Interface

- Chat interface
- Source transparency panel
- Session state memory
- Example prompts
- Conversation statistics

📂 Project Structure

```
project/
│
├── app.py                  # Streamlit UI
├── testing.py              # RAG logic (kafka_rag_answer)
├── ingest.py               # Text loading & DB creation
├── chroma_db/              # Persistent vector database
├── requirements.txt
└── README.md
```

⚙️ Installation

1️⃣ Clone the repository

```
git clone <repo-url>
cd kafka-rag
```

2️⃣ Create virtual environment

```
python -m venv lang_env
lang_env\Scripts\activate
```

3️⃣ Install dependencies

```
pip install -r requirements.txt
```

4️⃣ Install and run Ollama

Download Ollama and pull a model:

```
ollama pull llama3
```

Make sure Ollama is running.

🚀 Running the Application

```
streamlit run app.py
```

Open the browser at:

```
http://localhost:8501
```

📚 Metadata Design

Each chunk stored in ChromaDB includes metadata like:

```
{
    "author": "Franz Kafka",
    "work": "Metamorphosis",
    "type": "novel",
    "chunk_id": 12
}
```

This enables:

- Filtering by work
- Transparent source display
- Structured retrieval analysis

🧠 RAG Design Principles

- Responses must be grounded in retrieved passages
- No hallucinated content outside archive
- If context is insufficient → acknowledge limitation
- Persona styling layered *after* grounding

🛠️ Debugging Tips

Check stored works

```
collection.get(include=["metadatas"])
```

Check retrieved chunks

```
for doc in docs:
    print(doc.metadata)
    print(doc.page_content[:200])
```

If preview shows only '.'

- Verify ingestion pipeline
- Check chunking
- Ensure text was loaded correctly

🎯 Future Improvements

- Streaming token-by-token responses

- Similarity score display in UI
- Confidence score estimation
- Better metadata filtering logic
- Hybrid search (BM`25` + embeddings)
- Multi-work comparison mode

📊 Educational Value

This project demonstrates:

- Practical RAG implementation
- Vector database debugging
- Persona-conditioned generation
- Retrieval transparency design
- Streamlit conversational UI engineering

⚖️ Disclaimer

This system simulates Kafka’s literary voice for educational purposes.
Responses are generated by a language model and are not authentic historical writings.

👤 Author

Built as an advanced RAG experimentation project.

"I write differently from what I speak, I speak differently from what I think..."