Step-by-Step Guide: Building Your Own Generative AI (RAG) System

1. Define Your Use Case

- What questions should your AI answer?
- Who are your users?
- What data do you need (FAQs, manuals, reports, etc.)?

2. Collect and Prepare Data

- Gather documents (PDFs, web pages, text files, etc.)
- Clean and preprocess text (remove noise, fix encoding)
- Organize data into a consistent format (markdown, JSON, etc.)

3. Build a Knowledge Base

- Split documents into chunks (e.g., paragraphs, sections)
- Store each chunk with metadata (title, source, etc.)
- Save as markdown or JSON for easy processing

4. Generate Embeddings & Create a Vector Store

- Choose an embedding model (e.g., `sentence-transformers/all-MiniLM-L6-v2`)
- Convert each chunk to a vector (embedding)
- Store vectors in a vector database (e.g., FAISS, Chroma, Pinecone)

5. Integrate a Language Model (LLM)

- Choose your LLM: OpenAI (cloud), Ollama (local), HuggingFace, etc.
- Set up the LLM to answer questions using retrieved context
- Use frameworks like LangChain for easy integration

6. Build a Retrieval-Augmented Generation (RAG) Pipeline

- On each user question:
    1. Embed the question
    2. Retrieve top-k relevant chunks from the vector store
    3. Pass context + question to the LLM for answer generation

7. Expose Your System via API or Web Interface

- Use FastAPI or Flask to create a REST API
- Optionally, build a web chat interface (React, Streamlit, etc.)
- Log all interactions for monitoring and improvement

8. Add MLOps: Experiment Tracking & Data Versioning

- Use MLflow to track:
  - Latency, token usage, retrieval quality, etc.
  - Model versions and parameters
- Use DVC to version your data and knowledge base

9. Test and Evaluate

- Create a set of test questions
- Measure accuracy, latency, and user satisfaction
- Iterate on data, retrieval, and prompt engineering

10. Deploy and Monitor

- Deploy API/web app to cloud or on-premises
- Monitor health, usage, and errors
- Regularly update data and retrain embeddings as needed

Example Tech Stack

- **Data Processing:** Python, BeautifulSoup, Pandas
- **Embeddings:** sentence-transformers, HuggingFace
- **Vector Store:** FAISS, Chroma, Pinecone
- **LLM:** Ollama (Llama 3), OpenAI GPT, HuggingFace Transformers
- **API:** FastAPI, Flask
- **Web UI:** React, Streamlit, Flask templates
- **MLOps:** MLflow, DVC, Git
- **Deployment:** Docker, cloud VM, on-prem server

Sample Project Structure

```
project-root/
|-- data/
|   |-- raw/                # Source documents
|   `-- knowledge_base/     # Processed chunks/snippets
|-- scripts/                # All utility scripts
|-- app/                    # API and web app code
|-- models/                 # LLMs, embeddings
|-- mlflow.db               # MLflow tracking
|-- .dvc/                   # DVC config
|-- requirements.txt
`-- README.md
```

Tips for Success

- Start small: prototype with a few documents and questions
- Use open-source models for privacy and cost control
- Log everything: questions, answers, retrievals, errors
- Regularly update your knowledge base and embeddings
- Involve users for feedback and improvement

Resources

- LangChain Documentation
- MLflow Documentation
- DVC Documentation
- Ollama (local LLMs)
- HuggingFace Transformers
- FastAPI

Prepared by your AI coding assistant