

Production-Grade RAG Chatbot — End-to-End Guide

LangChain + Ollama Embeddings + Chroma Vector DB + Groq LLM + FastAPI + Web UI
+ Logging + LangSmith Tracing & Evaluation

Prepared for: Snazal Singh | Date: 18 Jan 2026

What you built

This document explains the complete system you implemented: a production-grade RAG (Retrieval-Augmented Generation) chatbot that can ingest documents, build a vector database, answer questions with citations, and provide observability (logging + tracing) and quality evaluation (LangSmith experiments).

- Document ingestion pipeline (PDF -> cleaned text -> chunking -> embeddings -> persistent vector DB).
- RAG pipeline (retrieve top-k chunks -> build prompt context -> Groq LLM answer -> citations).
- FastAPI backend providing /ask endpoint.
- Professional HTML/CSS chat UI.
- Production logging: retrieval latency, LLM latency, total latency, context length, retrieved chunk count.
- LangSmith tracing (call graph) and evaluation dashboard (experiments + custom evaluators).

1. RAG fundamentals

RAG stands for **Retrieval-Augmented Generation**. Instead of forcing an LLM to answer from its internal knowledge (which can hallucinate), RAG first retrieves relevant information from your documents and then generates an answer grounded in that context.

Core pipeline

- User question
- Retriever searches vector database (Chroma) for relevant chunks
- Top-k chunks are combined into a context block
- LLM (Groq) generates an answer using the context
- System returns answer + citations/sources

Why this matters

- Reduces hallucinations (answer must be supported by retrieved context).
- Allows Q&A; over private documents (resume, notes, PDFs).
- Makes system auditable using citations and tracing.

Key terms

Term	Meaning (simple)
Embedding	A numeric vector representing text meaning; used for similarity search.
Chunking	Splitting documents into smaller parts so retrieval is accurate and context fits in LLM prompt.
Vector DB	Database optimized to search embeddings using similarity (cosine / dot product).
Top-k	Number of chunks retrieved for each question.
Citations	Mapping answer back to source chunks/pages.

2. System architecture

You structured the project as modular components so ingestion, retrieval, API, and UI are separated. This is the same mindset used in production backends.

Folder structure

```
hanuman_god/
└── app.py
└── config.py
└── run_ingest.py
└── ingestion/
    ├── load_docs.py
    ├── clean_text.py
    ├── chunk_docs.py
    ├── build_vectorstore.py
    └── rag/
        ├── retriever.py
        ├── prompt.py
        ├── chain.py
        └── citations.py
    └── web/
        ├── templates/
        └── static/
    └── utils/
        └── logger.py
    └── data/
        └── raw_docs/
    └── chroma_db/
```

How data flows

- Place PDFs in data/raw_docs/
- Run run_ingest.py to build embeddings and store vectors in data/chroma_db/
- Start backend using FastAPI (uvicorn app:app --reload)
- UI calls /ask?q=... endpoint for chat

3. Ingestion pipeline (PDF -> Vector DB)

Ingestion converts raw documents into searchable vectors. In production, ingestion is critical: if chunking/cleaning is wrong, retrieval will fail even if the LLM is good.

Steps implemented

- **Load documents:** read PDFs from data/raw_docs/.
- **Clean text:** remove extra whitespace, fix encoding issues, normalize paragraphs.
- **Chunk documents:** RecursiveCharacterTextSplitter used to create chunks with overlap.
- **Create embeddings:** using OllamaEmbeddings with model **embeddinggemma:latest**.
- **Persist to Chroma:** stored in data/chroma_db/ to enable fast retrieval.

Chunking parameters (important)

- **chunk_size:** max characters per chunk (e.g., 800-1200).
- **chunk_overlap:** overlapping characters between chunks (e.g., 80-150).
- Overlap prevents context from breaking across chunk boundaries.

Command used

```
python run_ingest.py
```

4. Retrieval + Generation (RAG pipeline)

The RAG pipeline is the runtime system that answers user questions.

Retriever

A retriever wraps the vector database and returns the most similar chunks for the query.

```
docs = retriever.invoke(question)
docs = docs[:FINAL_K]
```

Prompt + context

Retrieved chunks are formatted into a single context block with source identifiers to support citations.

LLM call (Groq)

Groq provides extremely fast inference. LangChain is used to call Groq models via ChatGroq.

Citations

- Each chunk has metadata (source doc name, page, chunk index).
- Citations are returned alongside the answer to make the response auditable.

5. FastAPI backend

FastAPI exposes your RAG as a real API product. This makes it easy to integrate with web/mobile apps.

Key endpoint

```
GET /ask?q=...
```

Testing

- Browser: <http://127.0.0.1:8000/ask?q=what%20is%20resume>
- Swagger UI: <http://127.0.0.1:8000/docs>
- Curl:

```
curl "http://127.0.0.1:8000/ask?q=what%20is%20resume"
```

What 200 OK means

A 200 response confirms the complete RAG pipeline works end-to-end: request -> retrieval -> LLM -> response.

6. Web UI (professional chatbot)

You created a simple, professional UI using HTML/CSS. The UI sends the user query to the FastAPI endpoint and renders the answer and sources.

Recommended UI features

- Chat bubbles (user vs assistant)
- Typing indicator / loader
- Source panel (expand/collapse citations)
- Error banner for API failures
- Mobile responsive layout

7. Production logging (observability)

Logging is essential in production to measure performance and debug failures. You implemented structured logs in the RAG chain.

Metrics logged

- **Retrieved chunk count:** number of docs returned by the retriever.
- **Retrieval latency:** time taken for Chroma similarity search.
- **Context length:** total size of context passed to the LLM.
- **LLM latency:** time taken by Groq model to generate response.
- **Total latency:** end-to-end time for the request.

Example log output

```
[2026-01-18 05:19:45,228] [INFO] [rag_chain] RETRIEVED_DOCS: 6 in 0.28s
[2026-01-18 05:19:45,229] [INFO] [rag_chain] CONTEXT_LENGTH_CHARS: 3046
[2026-01-18 05:19:46,277] [INFO] [rag_chain] LLM_LATENCY: 1.05s
[2026-01-18 05:19:46,278] [INFO] [rag_chain] TOTAL_LATENCY: 1.66s
```

Interpretation

- 0.28s retrieval indicates vector DB search is fast.
- 1.05s LLM latency is normal and production-acceptable (network + token generation).
- Total 1.66s end-to-end latency is strong for RAG.

8. Tracing with LangSmith

Tracing goes beyond logging. While logs are individual messages, traces capture the complete timeline of each request - retriever calls, prompt construction, and LLM generation - as a graph.

How you enabled tracing

- Added environment variables in .env:
 - LANGCHAIN_TRACING_V2=true
 - LANGCHAIN_API_KEY=...
 - LANGCHAIN_PROJECT=hanuman_god_rag

Function-level tracing

```
from langsmith import traceable

@traceable(name="hanuman_rag_answer")
def answer_question(question: str):
    ...
```

What the trace dashboard shows

- Call graph: hanuman_rag_answer -> retriever -> prompt -> Groq LLM
- Inputs/outputs at each step (query, retrieved docs, prompt text, answer)
- Latency per step
- Errors with stack traces

9. Evaluation (LangSmith Experiments)

Evaluation measures RAG quality - not just speed. You created a dataset and ran experiments using custom evaluators (answer relevance, conciseness, hallucination).

Why evaluation matters

- Ensures answers are relevant to questions.
- Detects hallucinations (especially on trick questions).
- Allows regression testing: compare model/prompt/retrieval versions.

Dataset design (resume example)

- Real questions: name, MCA year, projects, skills.
- Trick questions: DOB, salary, father name (should return 'Not mentioned').

Key scores explained

- **Answer relevance:** answer is on-topic.
- **Conciseness:** answer is clear and not unnecessarily long.
- **Hallucination:** answer does not invent facts not supported by documents.

Appendix A: Common errors you solved

During development you encountered real-world issues typical in production builds. Documenting these is valuable for interviews.

1) ModuleNotFoundError: ingestion / rag

- Cause: running scripts from subfolders changes Python path.
- Fix: run as module: `python -m eval.manual_eval` or add `__init__.py`.

2) LangChain import changes

- Some modules moved between langchain and langchain_community.
- Fix: pin versions and use correct imports.

3) Ollama model not found (404)

- Cause: wrong embedding model name.
- Fix: verify model using `ollama list` and update EMBEDDING_MODEL.

4) Groq 401 invalid key

- Cause: wrong GROQ_API_KEY in .env.
- Fix: set GROQ_API_KEY correctly and restart server.

5) Groq model decommissioned

- Cause: model name no longer supported.
- Fix: switch to active models (e.g., llama-3.1-8b-instant).

Appendix B: What makes this 'production-grade MVP'

Your system is an end-to-end production-grade MVP because it has a complete pipeline plus observability and evaluation. To reach full enterprise product level, add authentication, multi-user document separation, cloud deployment, background ingestion jobs, and stronger security guardrails.

Recommended next upgrades

- Hybrid retrieval (BM25 + vector).
- Reranking (cross-encoder).
- Prompt injection defense + strict refusal policy.
- PDF upload + document management UI.
- Docker + cloud deployment (Render/Railway/Fly.io).
- CI evaluation regression tests using LangSmith.

Appendix C: Screenshots (reference)

You can paste/replace these screenshots in your final YouTube/PDF notes. They illustrate evaluator options and experiment dashboards.

2.21.05 PM

The screenshot shows a dark-themed sidebar menu with the following items:

- Correctness LLM as judge** >
Whether an answer semantically matches ...
- Hallucination LLM as judge** >
Whether an answer hallucinates facts.
- Conciseness LLM as judge** >
Whether an answer is a concise response...
- Code Checker LLM as judge** >
Whether code produced solves the questi...
- Toxicity LLM as judge** >
Whether given content contains toxic ele...
- Answer Relevance LLM as judge** >
Whether an output answer is relevant to a...
- Plan Adherence LLM as judge** >
Whether an agent followed a declared pla...

2.24.07 PM

Evaluator Name

Correctness

Prompt & Model

Create your evaluator by defining the criteria you want to evaluate in your prompt. Create your prompt below or use a saved prompt from Prompt Hub.

Use hub prompt

Preview OpenAI gpt-5-mini

Start typing {{ to create a variable.

- Check for factual accuracy and completeness
- Focus on correctness of information rather than style or verbosity

```
</Instructions>
<Reminder>
The goal is to evaluate factual correctness and completeness of the response.
</Reminder>
```

```
<input>
  
</input>
```

Example

Use this Example from your dataset to help map the placeholder variables in the evaluation prompt to the inputs, outputs and references outputs.

INPUT

{}

REFERENCE OUTPUT

{}

OUTPUT

Run an experiment [via the SDK](#) or [in the Playground](#) to generate outputs. This is recommended before creating an evaluator to aid

2.24.19 PM

Evaluator Name

Correctness

Prompt & Model

Use the reference outputs below to help you evaluate the correctness of the response:

```
<output>
</output>
```

```
<reference_outputs>
  
</reference_outputs>
```

+ Message

Feedback configuration

Define your evaluation criteria. Describe what you're measuring, then select a response format. This configuration structures how your evaluation results are returned.

correctness Include reasoning

Description

TRUE if the output is factually accurate, complete, logically consistent, and matches the reference output in key information. FALSE if there are any factual errors, missing information, logical inconsistencies, or significant deviations from the reference output.

2.27.22 PM

New Dataset

Name: ds-charming-sigh-34

Description: Type description...

Application: No applications available

Input fields: question

Output fields: expected

Metadata fields:

Dataset schema (optional) Infer schemas from file

Attached to the example. What is my full name exactly as written in the resume? Must match name exactly from resume.

Inputs: Question

Outputs: Expected

Metadata:

2.50.22 PM

LangSmith Personal > Playground

Playground Iterate on and test prompts.

Prompts: Select prompt HUMAN + Message + Output Schema + Tool

Secrets & API keys: OPENAI_API_KEY

New Experiment: 14 Examples resume_eval_set

#	Reference Outputs	Outputs
1	ick question: What is my ther's name?	expected Not mentioned in resume Press Start to run your prompt over the dataset
2	hat are my strengths based the resume? Give 3 rengths.	expected 3 strengths Press Start to run your prompt over the dataset
3	hat is my email address and one number mentioned in	expected Email + phone exactly as written Press Start to run your prompt over the dataset

3.05.23 PM

