# RAG Internship Project — Usage Guide

## 1. Project Overview

**RAG (Retrieval-Augmented Generation)** is a lightweight demo that combines **document retrieval with AI-powered generation**.

- **Goal:** Allow users to query a dataset and get AI-assisted answers based on relevant document chunks.
- **Tech Stack:** Python, Streamlit, FAISS, Sentence Transformers, OpenAI GPT.

**Project Structure:**

RAG_AI_INTERNSHIP_PROJECT/

│

├– app/ # Streamlit app

├– data/ # Raw documents, processed docs, FAISS index

├– scripts/ # Ingest and embedding builder scripts

├– utils/ # Helper functions

├– .venv/ # Python virtual environment

├– requirements.txt # Dependencies

├– README.md

├– .env # OpenAI API key

---

## 2. Installation

**Step 1 — Clone the repository**

git clone https://github.com/SudheerKonduboina/RAG_AI_INTERNSHIP_DEMO.git

cd RAG_AI_INTERNSHIP_DEMO

**Step 2 — Create virtual environment**

python -m venv .venv

**Step 3 — Activate virtual environment**

- **Windows:**

.\\.venv\\Scripts\\activate

- **macOS/Linux:**

source .venv/bin/activate

**Step 4 — Install dependencies**

pip install -r requirements.txt

**Explanation:** This installs all required packages: streamlit, sentence-transformers, faiss-cpu, openai, python-dotenv.

---

## 3. Setting Up Environment Variables

Create a .env file in the project root:

OPENAI_API_KEY=sk-your_openai_key_here

**Explanation:** The OpenAI key allows GPT to generate responses. Never share .env on GitHub.

---

## 4. Preparing Data

**Step 1 — Place raw documents**

- Create folder: data/raw_docs/
- Place your PDFs or text files there.
- Example: data/raw_docs/Full_AI_Content_All_Lines.pdf

**Step 2 — Ingest documents**

python -m scripts.ingest --source data/raw_docs --out data/processed_docs.json

**Explanation:**

- ingest.py reads raw documents.
- It splits content into smaller chunks and saves them in processed_docs.json.

---

## 5. Building Embeddings & FAISS Index

python -m scripts.build_embeddings --docs data/processed_docs.json --index_path data/faiss_index

**Explanation:**

- Converts text chunks into vector embeddings using SentenceTransformer.
- Saves embeddings to FAISS index (faiss.index) for fast retrieval.
- Also stores chunks.pkl (text chunks) and ids.pkl (chunk identifiers).

---

## 6. Running the Streamlit App

streamlit run app/streamlit_app.py

**Explanation:**

- Opens a web UI for asking questions about your dataset.
- Uses FAISS to find relevant chunks.
- Optionally uses OpenAI GPT to generate answers.
- Displays retrieval and generation time.

---

## 7. How the Project Works (Step by Step)

### 7.1 Retrieval

1. User inputs a query in Streamlit UI.
2. The retrieve() function generates a vector embedding for the query.
3. FAISS searches top-k most similar chunks.
4. Retrieved chunks are returned.

### 7.2 Prompt Composition

1. Retrieved chunks are formatted into a prompt with context.
2. The system ensures **answers are based only on context**.
3. Default message if answer not found: "The answer is not available in the provided documents."

### 7.3 AI Generation

1. If OpenAI API key is set, openai.chat.completions.create() is called.
2. GPT model (gpt-4o-mini) generates a response.
3. Streamlit displays the answer and retrieved chunks.

---

## 8. Scripts Explained

| Script | Purpose |
|---|---|
| scripts/ingest.py | Reads raw documents, splits them into chunks, saves processed_docs.json |
| scripts/build_embeddings.py | Builds vector embeddings, stores FAISS index + chunk metadata |
| scripts/eval.py | Optional: Evaluate retrieval or generation accuracy |
| app/streamlit_app.py | Web interface for RAG Q&A |

---

## 9. Notes and Troubleshooting

- **FAISS index not found:** Run build_embeddings.py first.
- **OpenAI Error 401:** Check .env key is correct.
- **Quota exceeded:** Upgrade OpenAI plan.
- **Permissions issues:** Make sure data/ and .venv/ folders have read/write access.

---

## 10. Example Usage

1. Run Streamlit:

streamlit run app/streamlit_app.py

2. Input:

What is AI?

3. Output:

- Retrieved chunks (top 5 relevant).
- Generated answer using GPT.
- Retrieval & generation time displayed.

---

## 11. Recommended Improvements

- Add **FastAPI** endpoint for programmatic access.
- Store embeddings in **persistent cloud storage** for larger datasets.
- Fine-tune **GPT model** for domain-specific questions.
- Add **logging & error handling** for robust production use.

---

## 12. License



---

**Author:** SUDHEER KONDUBOINA