AI Document Question-Answering Assistant (RAG)

Proof Document (Work Samples, Results, and Evidence)

Date: 07 Jan 2026

GitHub Repository: https://github.com/vamsireddy99126/ai-document-qa-rag

1. Project Summary

A web application that lets a user upload documents (PDF/TXT/DOCX), ask questions in natural language, and receive answers grounded strictly in the uploaded documents. The system uses Retrieval-Augmented Generation (RAG): document chunking, embeddings, vector search, and context injection to reduce hallucinations.

Core user flow: Upload → Index → Ask → Retrieve → Answer + Sources

2. System Architecture

• Document ingestion (PDF/TXT/DOCX) using loaders
• Chunking using RecursiveCharacterTextSplitter
• Embeddings (supports local free embeddings via SentenceTransformers; OpenAI-ready)
• Vector store using FAISS for similarity search
• Retriever to fetch top-k relevant chunks
• Answer generation function with strict grounding rules + source display
• Streamlit UI for upload, indexing, and chat

Architecture map (modules and responsibilities):

User	Upload document
Loader	Extract text
Splitter	Chunk text (overlap)
Embeddings	Convert chunks to vectors
FAISS	Store vectors + similarity search
Retriever	Top-k relevant chunks
Answerer	Answer using ONLY retrieved context
UI	Show answer + source chunks

3. Tools, Software, and Materials Used

Category	Tool / Software	Purpose
Language	Python 3.11	Core implementation language
UI	Streamlit	Upload docs, build index, chat interface
RAG framework	LangChain (modular)	Loaders, splitting utilities, vector store integration
Vector DB	FAISS	Fast similarity search over embeddings
Embeddings	SentenceTransformers (all-MiniLM-L6-v2)	Free local embeddings (no API quota)
Docs	pypdf, docx2txt	Document text extraction
Config	python-dotenv (.env)	Manage secrets and environment switches
Dev tools	Git + GitHub	Version control + portfolio proof

4. Skills Demonstrated (Recruiter-Friendly)

• RAG fundamentals: embeddings + retrieval + context injection
• Vector similarity search and top-k retrieval
• Document ingestion pipelines (PDF/TXT/DOCX) and text preprocessing
• Prompt grounding / hallucination reduction (answer only from provided context)
• Streamlit app development (upload, controls, chat UI, session state)
• Debugging and dependency management (LangChain modular changes, env setup)
• Git workflow and portfolio packaging (README, clean repo, reproducible runs)

5. Evidence Checklist (Add these to strengthen proof)

• Screenshot: Streamlit app home screen and file uploader
• Screenshot: After clicking 'Build Index' showing success message
• Screenshot: Example Q&A; with Sources expanded (show chunks match the PDF)
• Terminal output: python --version (3.11), pip install, and 'python -m streamlit run app.py'
• GitHub screenshot: repo tree (app.py, src/, README.md) and commits history
• Optional: short screen recording (30-60s) demonstrating upload → question → answer

6. How to Run (Copy/Paste)

• git clone https://github.com/vamsireddy99126/ai-document-qa-rag.git
• cd ai-document-qa-rag
• python3.11 -m venv venv
• source venv/bin/activate
• pip install -r requirements.txt
• python -m streamlit run app.py

7. Notes on API Quota and Free Mode

If OpenAI quota is unavailable, the project runs in free mode using local embeddings (SentenceTransformers). To avoid accidental API calls, set **EMBEDDINGS_PROVIDER=local** in .env or remove OPENAI_API_KEY.