Agentic RAG Chatbot for Multi-Format Document QA using MCP

- Goal: Build a chatbot to answer questions using uploaded docs (PDF, DOCX, CSV, PPTX, TXT, MD).
- Agent-based architecture with message-passing (MCP).
- RAG: Ingest, Retrieve, Generate answer using context.
- Frontend built using Streamlit.

Agent-Based Architecture with MCP

- CoordinatorAgent: Orchestrates all agent interactions.
- IngestionAgent: Parses, chunks, embeds docs into vector store.
- RetrievalAgent: Retrieves top-k relevant chunks using FAISS.
- LLMResponseAgent: Uses retrieved context to generate final answer.
- Messages exchanged using structured JSON (MCP format).

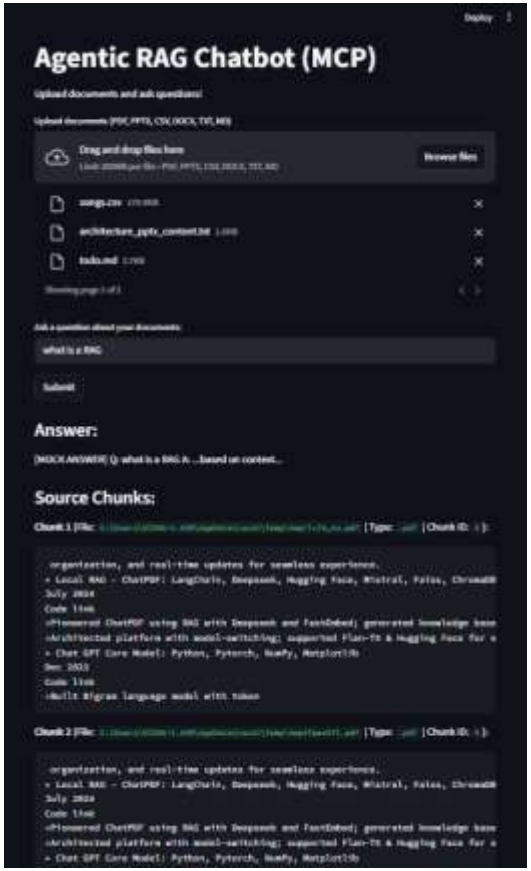
System Flow Diagram (Message Flow)

- UI → CoordinatorAgent: USER_QUERY
- CoordinatorAgent → IngestionAgent: INGEST_REQUEST
- IngestionAgent → RetrievalAgent: RETRIEVE_REQUEST
- RetrievalAgent → LLMResponseAgent: CONTEXT_RESPONSE
- LLMResponseAgent → CoordinatorAgent → UI: ANSWER
- Trace_id used for tracking query lifecycle

Tech Stack Used

- Frontend: Streamlit
- Embeddings: SentenceTransformers (all-MiniLM-L6-v2)
- Vector DB: FAISS
- LLM Backend: OpenAI GPT-4 / HuggingFace (mock fallback)
- Parsing: PyMuPDF, python-docx, pandas, pptx, markdown
- MCP Protocol: Custom in-memory JSON structure

📸 UI Screenshot



Challenges Faced & Future Scope

- Challenges: Parsing table-heavy documents, designing reusable MCP, chunking efficiency.
- Future Scope:
- - Add web/audio agents
- - Use Redis or Kafka for distributed MCP
- - Token streaming responses
- - Stateful multi-turn memory via trace_id