# Agentic RAG Chatbot for Multi-Format Doc QA using Model Context Protocol (MCP)

Akash Subramanian

## Problem Statement

The goal is to build an agent-based Retrieval-Augmented Generation (RAG) chatbot that can answer user queries using uploaded documents of various formats (PDF, DOCX, PPTX, CSV, TXT, MD). The system must follow an agentic structure with modular responsibilities and use a Model Context Protocol (MCP) for all inter-agent communication and LLM interfacing.

## Solution Overview

We propose a modular architecture consisting of three main agents:

- **IngestionAgent** — responsible for parsing and chunking uploaded files.
- **RetrievalAgent** — handles embedding and semantic search over document chunks.
- **LLMResponseAgent** — constructs context-aware prompts and interacts with the local LLM (via Ollama).

Each agent communicates using structured JSON-like messages based on the Model Context Protocol (MCP), enabling decoupled and extensible agent behavior.

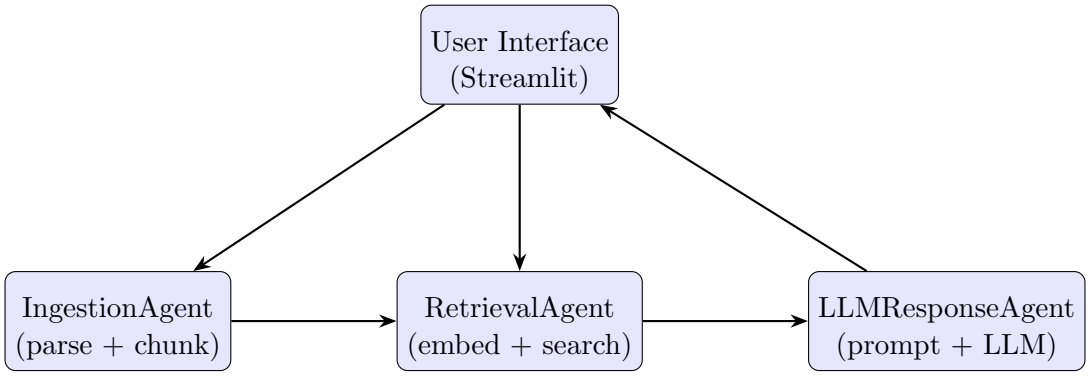
## Workflow

1. **User uploads documents** (e.g., `sales_review.pdf`, `metrics.csv`).
2. **UI sends MCP message** to `IngestionAgent`.
3. `IngestionAgent` parses the files, chunks text, and sends to `RetrievalAgent`.
4. **User submits a query** like: *"What KPIs were tracked in Q1?"*
5. `RetrievalAgent` embeds the query and retrieves top-k chunks.
6. `RetrievalAgent` sends context to `LLMResponseAgent`.
7. `LLMResponseAgent` builds a prompt and queries a local model via Ollama.
8. Answer is routed back to UI and displayed in context.

## MCP Message Format Example

```
{
  "sender": "RetrievalAgent",
  "receiver": "LLMResponseAgent",
  "type": "RETRIEVAL_RESULT",
  "trace_id": "rag-457",
  "payload": {
    "retrieved_context": [
      "slide 3: revenue up",
      "doc: Q1 summary..."
    ],
    "query": "What KPIs were tracked in Q1?"
  }
}
```

## System Architecture Diagram




## Tech Stack

| **Component** | **Technology / Tool** |
|---|---|
| Frontend UI | Streamlit (interactive web app) |
| File Parsing | PyMuPDF (PDF), python-docx (DOCX), python-pptx (PPTX), pandas (CSV), open() for TXT/MD |
| Chunking Strategy | RecursiveCharacterTextSplitter (LangChain or custom) |
| Embeddings | HuggingFace Sentence Transformers (e.g., `all-MiniLM`) |
| Vector Store | FAISS (lightweight, local) |
| LLM Backend | Ollama (local serving of models like `gemma`, `mistral`) |
| Agent Messaging | Custom MCP (Model Context Protocol) using Python classes |

## Why This Approach?

- **Agentic Architecture:** Allows modular testing, debugging, and scaling of each task (parse, retrieve, respond).
- **MCP Protocol:** Creates a clean, traceable way for agents to communicate, making debugging and orchestration easier.
- **Multi-Format Support:** Widens usability across different business/report formats.

- **Offline Free:** By using Ollama and local models, no OpenAI keys or internet dependency is needed.
- **Streamlit UI:** Simple, fast to build, and fully supports interactive user queries.