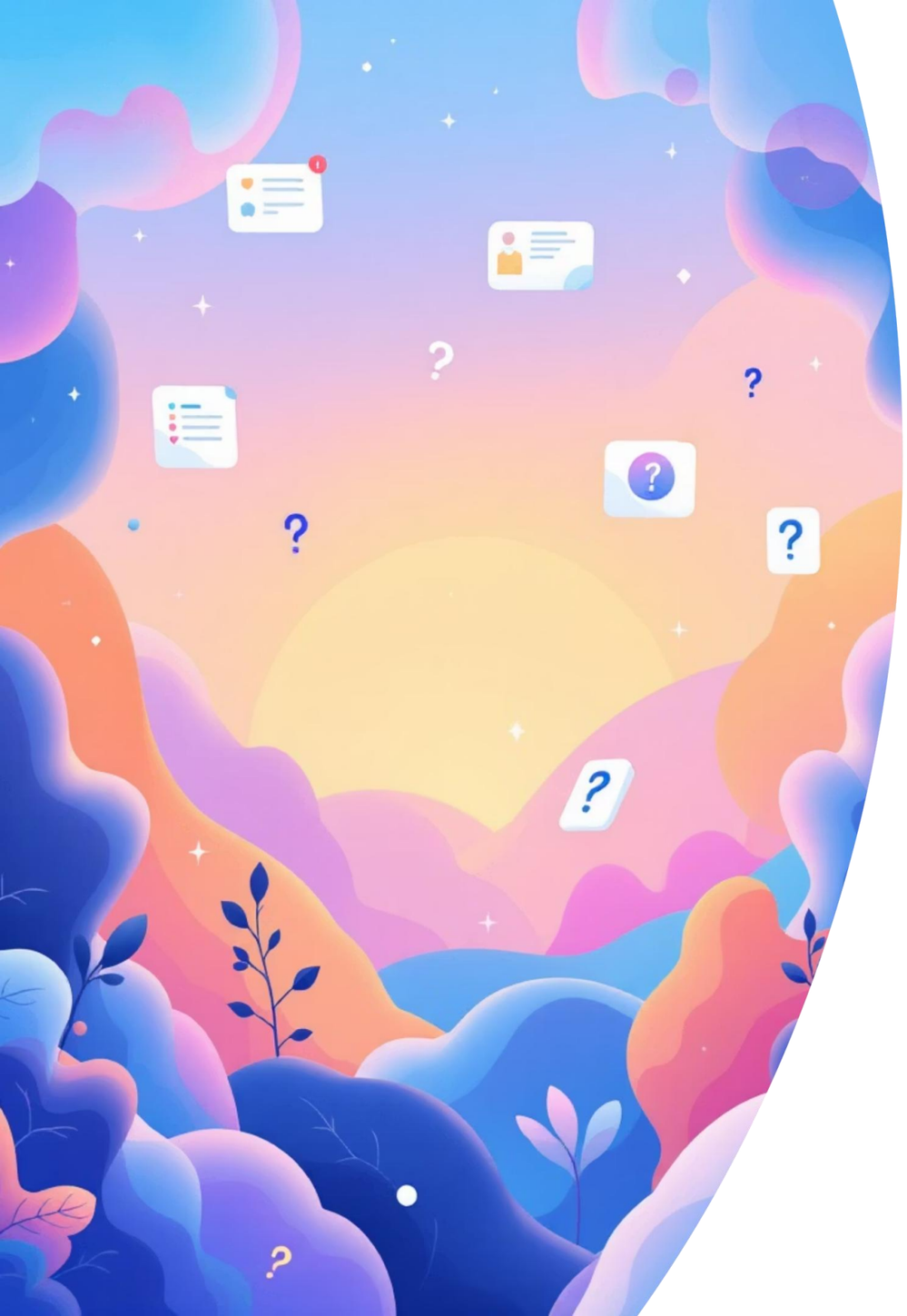


Agentic RAG Chatbot for Multi-Format Document QA using MCP

Coding Round Project Submission

Technical Implementation for AI/ML Systems Role

October 2025

N.Sushanth

Problem Statement & Requirements

Multi-Format Support

Build an intelligent RAG chatbot capable of processing and answering queries from uploaded documents across multiple file formats: PDF, DOCX, PPTX, CSV, and TXT files.

Agentic Architecture

Implement a distributed agent system using Model Context Protocol (MCP) for structured communication between specialized components handling document ingestion, retrieval, and response generation.

Conversational Intelligence

Deliver context-aware, conversational responses that maintain coherence across multi-turn conversations while providing source attribution for transparency and verification.

System Architecture Overview

Frontend Layer

Streamlit-based user interface providing intuitive file upload capabilities and real-time chat interaction. The interface maintains conversation history and displays source attributions for each response.

Backend Services

FastAPI-powered backend orchestrating three specialized agents:

- **IngestionAgent** - Parses documents and generates embeddings
- **RetrievalAgent** - Performs semantic search with reranking
- **LLMResponseAgent** - Generates contextual responses using Gemini

Data Infrastructure

ChromaDB serves as the persistent vector store, while Google Gemini 2.5 Pro handles both embedding generation and response synthesis. MCP ensures structured, type-safe communication between all components.

MCP Message Flow & Communication Protocol

Document Upload

User uploads documents through Streamlit UI, triggering MCP message to IngestionAgent for processing and embedding generation.

Vector Storage

Documents are parsed, chunked, and embedded using Gemini embedding model, then stored persistently in ChromaDB with metadata.

Query Processing

CoordinatorAgent receives user question and sends structured MCP message to RetrievalAgent for semantic search and context retrieval.

Response Generation

LLMResponseAgent receives context via MCP CONTEXT_RESPONSE message and generates final answer using Gemini with source attribution.

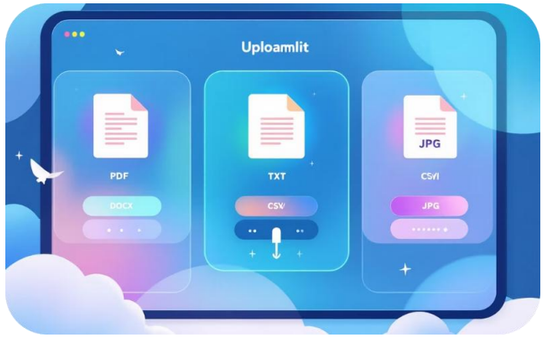
Example MCP Message Structure

```
{ "sender": "RetrievalAgent", "receiver": "LLMResponseAgent",  "type": "CONTEXT_RESPONSE", "payload": { "query": "What KPIs were tracked?", "context": ["Slide 3: revenue growth...", "Doc: Q1 summary..."] }}
```

Technology Stack & Implementation Details

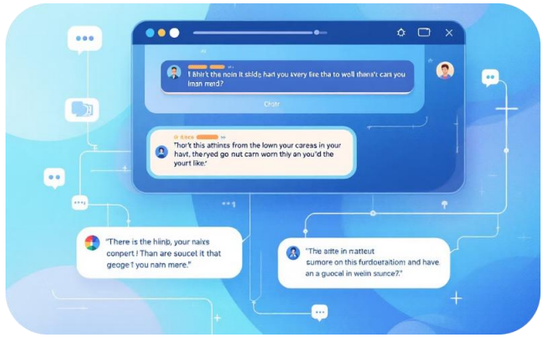
Layer	Technologies & Tools
Frontend	Streamlit for rapid UI development with file upload and chat components
Backend	FastAPI for high-performance async API endpoints and request handling
Agents	Python with LangChain for document processing, Pydantic for type validation
Vector DB	ChromaDB for persistent vector storage with metadata filtering capabilities
LLM	Google Gemini 2.5 Pro API for both embeddings and text generation
Protocol	Custom MCP implementation using Pydantic schemas for type-safe messaging

User Interface & Interaction Flow



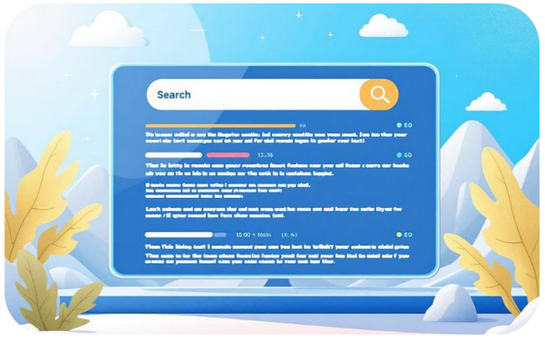
Document Upload Interface

Intuitive sidebar allowing users to upload multiple documents simultaneously. Progress indicators show processing status and successful ingestion confirmation.



Interactive Chat Experience

Real-time conversation interface displaying AI responses with clear source attributions. Users can ask follow-up questions while maintaining conversation context.



Contextual Response Display

Each response includes relevant document excerpts, source file names, and confidence indicators, enabling users to verify information accuracy and explore source materials.

Technical Challenges & Engineering Solutions

Multi-Format Document Parsing

Challenge: Handling diverse document formats with varying structures and metadata requirements across PDF, DOCX, PPTX, CSV, and TXT files.

Solution: Implemented LangChain community loaders with format-specific preprocessing pipelines. Each loader extracts text while preserving document structure and metadata for enhanced retrieval accuracy.

Message Protocol Consistency

Challenge: Ensuring reliable, type-safe communication between distributed agents while maintaining message integrity and handling edge cases.

Solution: Developed Pydantic-based MCP schemas with comprehensive validation rules. Implemented retry mechanisms and error handling for robust inter-agent communication.

Retrieval Accuracy Optimization

Challenge: Achieving high precision in semantic search results while balancing relevance and diversity in retrieved document chunks.

Solution: Combined multi-query expansion with advanced reranking algorithms. Implemented hybrid search combining semantic similarity with keyword matching for improved accuracy.

Multi-Turn Conversation Handling

Challenge: Maintaining conversation context across multiple exchanges while handling follow-up questions and clarifications effectively.

Solution: Implemented conversation history condensation using LLM-powered question reformulation. System maintains context window while optimizing for token efficiency.

Future Enhancements & Project Deliverables

Planned Improvements

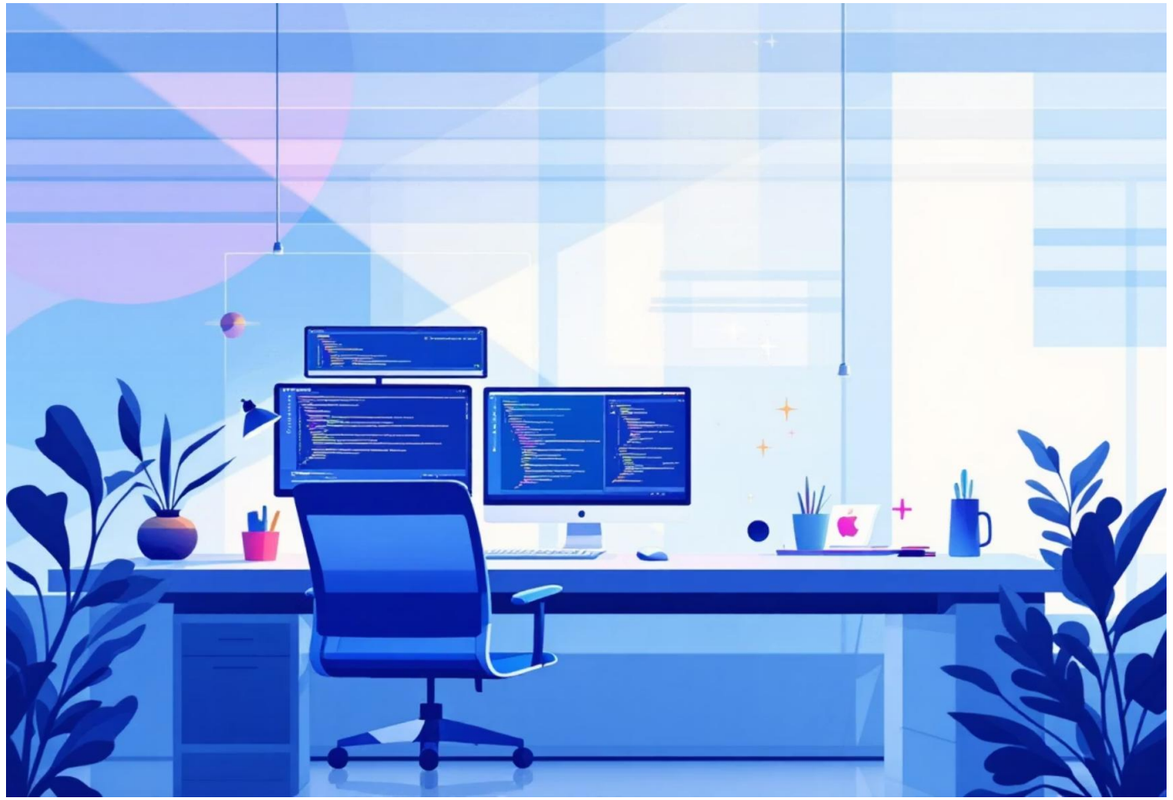
- **Open-Source LLM Integration:** Replace Gemini with Mistral or Llama 3 for cost optimization and enhanced privacy
- **User Management System:** Implement authentication, user sessions, and document access controls
- **Performance Optimization:** Add Redis caching layer for faster retrieval and response times
- **Production Deployment:** Deploy on Hugging Face Spaces, Render, or AWS with auto-scaling capabilities

Additional Features

- Real-time collaboration and document sharing
- Advanced analytics and usage metrics
- Custom embedding fine-tuning for domain-specific knowledge

Complete Deliverables

- Comprehensive GitHub repository with full source code
- Detailed README with setup and deployment instructions
- Technical presentation (this document)
- Optional 5-minute demonstration video showcasing architecture, code walkthrough, and live application demo



Project demonstrates advanced understanding of agentic architectures, RAG implementation, and production-ready system design patterns essential for AI/ML systems engineering roles.