Intelligent RAG System - Agent-based Architecture Analysis

🏗️ Agent-based Architecture with MCP Integration

System Overview

This intelligent RAG (Retrieval-Augmented Generation) system implements a multi-agent architecture using the Model Context Protocol (MCP) for inter-agent communication. The system processes documents, creates embeddings, and provides intelligent responses through coordinated agent interactions.

🤖 Agent Architecture

1. CoordinatorAgent (Orchestrator)

- **Role: Main workflow orchestrator and UI interface**
- **Responsibilities:**
  - **Manages document upload requests**
  - **Coordinates query processing**
  - **Handles system-wide operations (clear index)**
  - **Maintains active trace tracking**
  - **Updates UI state based on agent responses**

2. IngestionAgent (Document Processor)

- **Role: Document parsing and preprocessing**
- **Responsibilities:**
  - **Processes multiple file formats (PDF, DOCX, PPTX, CSV, TXT)**
  - **Extracts text content from documents**
  - **Chunks text with configurable overlap (250 words, 50 overlap)**
  - **Validates extracted content quality**
  - **Sends processed chunks to RetrievalAgent**

3. RetrievalAgent (Embedding & Search)

- **Role: Semantic search and retrieval**
- **Responsibilities:**
  - **Creates embeddings using SentenceTransformer**
  - **Manages FAISS vector index**
  - **Performs semantic similarity search**
  - **Maintains chunk metadata with source tracking**
  - **Handles index clearing operations**

4. LLMResponseAgent (Response Generation)

- **Role: Context-aware response generation**
- **Responsibilities:**
  - **Integrates with Google Gemini for LLM responses**
  - **Synthesizes answers from retrieved context**
  - **Implements fallback rule-based responses**
  - **Provides source attribution**
  - **Handles different query patterns (requirements, citations, definitions, etc.)**

🔄 Message Flow Architecture

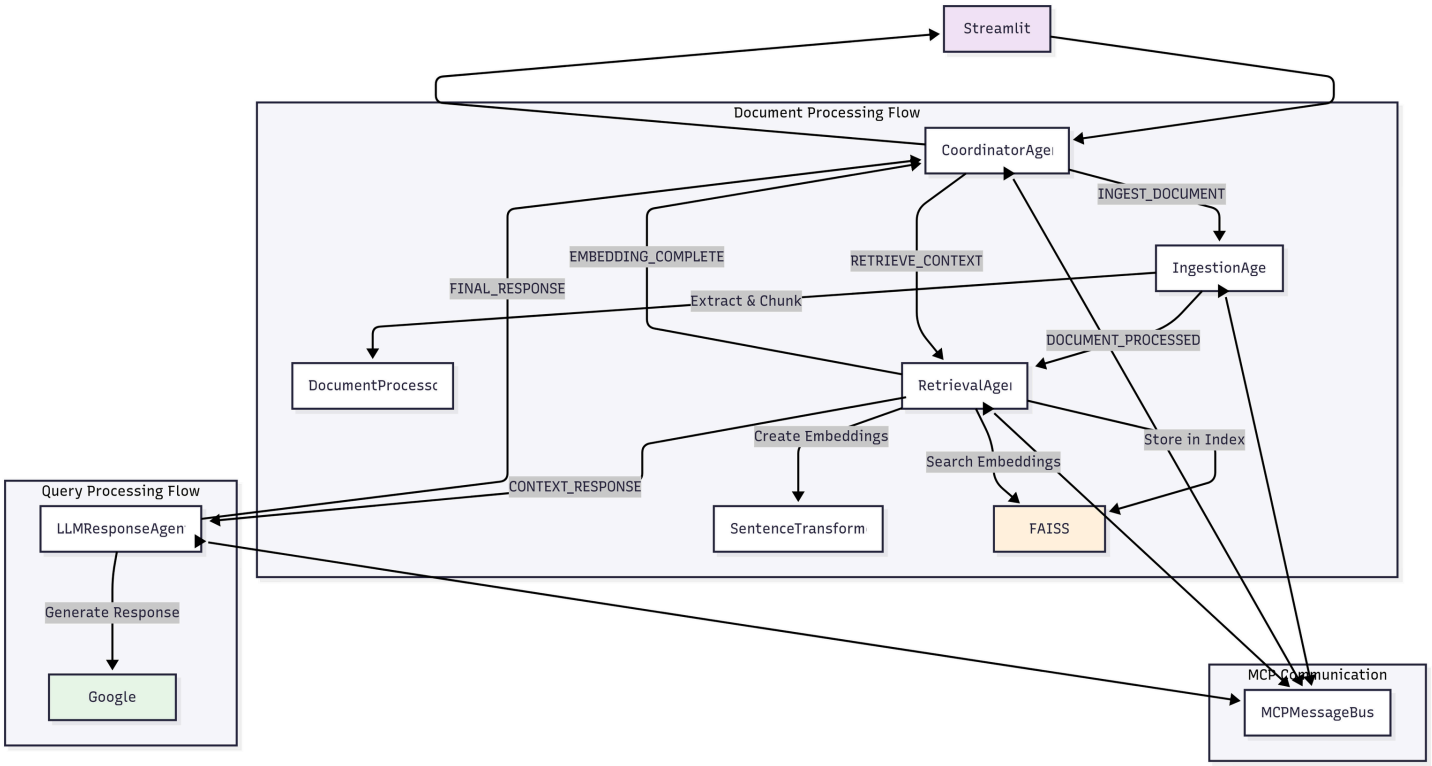
MCP Message Structure

```
@dataclass
class MCPMessage:
    sender: str        # Agent identifier
    receiver: str      # Target agent
    type: str          # Message type
    trace_id: str      # Request tracking
    payload: Dict      # Message data
    timestamp: str     # Message timestamp
```

Message Types

- **INGEST_DOCUMENT: Document processing request**
- **DOCUMENT_PROCESSED: Processed document with chunks**
- **RETRIEVE_CONTEXT: Context retrieval request**
- **CONTEXT_RESPONSE: Retrieved context with chunks**
- **FINAL_RESPONSE: Generated response with sources**
- **ERROR: Error handling**
- **EMBEDDING_COMPLETE: Embedding completion notification**
- **CLEAR_INDEX: Index clearing request**

📊 System Flow Diagram



🛠️ Tech Stack

Core Framework

- **Streamlit: Web UI framework for interactive interface**
- **Python 3.8+: Primary programming language**
- **asyncio: Asynchronous programming for agent coordination**

Document Processing

- **PyPDF2: PDF text extraction**
- **python-docx: Word document processing**
- **python-pptx: PowerPoint presentation processing**
- **pandas: CSV data processing and analysis**

Machine Learning & Search

- **SentenceTransformers: Text embeddings (all-MiniLM-L6-v2)**
- **FAISS: Vector similarity search and indexing**
- **NumPy: Numerical operations for embeddings**

LLM Integration

- **Google Generative AI: Gemini model integration**
- **Custom fallback system: Rule-based responses when LLM unavailable**

Architecture Patterns

- **Model Context Protocol (MCP): Inter-agent communication**
- **Publish-Subscribe Pattern: Message bus architecture**
- **Async/Await Pattern: Non-blocking operations**
- **Dataclass Pattern: Structured message formatting**

🎯 Key Features

1. Multi-format Document Support

- **PDF, DOCX, PPTX, CSV, TXT, Markdown**
- **Intelligent text extraction per format**
- **Content validation and quality checks**

2. Semantic Search

- **Vector embeddings with SentenceTransformer**
- **FAISS-powered similarity search**
- **Context-aware chunk retrieval**

3. Intelligent Response Generation

- **LLM integration with Google Gemini**
- **Pattern-based query handling**
- **Source attribution and citation**
- **Fallback rule-based responses**

4. Real-time Communication

- **MCP message bus for agent coordination**
- **Trace-based request tracking**
- **Async message handling**
- **Real-time UI updates**

5. Robust Error Handling

- **Comprehensive error propagation**
- **Graceful degradation**
- **User-friendly error messages**
- **System state recovery**

🚀 Workflow Process

Document Upload & Processing

1. **User uploads document via Streamlit UI**
2. **CoordinatorAgent receives file and creates trace**
3. **IngestionAgent processes document and extracts text**
4. **Text is chunked with overlap for better context**

5. **RetrievalAgent creates embeddings and stores in FAISS**
6. **System notifies completion and updates UI**

Query Processing

1. **User submits query through chat interface**
2. **CoordinatorAgent initiates retrieval process**
3. **RetrievalAgent searches for relevant chunks**
4. **LLMResponseAgent generates contextual response**
5. **Response is delivered with source attribution**
6. **Chat history is updated in real-time**

System Management

- **Index clearing functionality**
- **Document tracking and status**
- **MCP message logging**
- **Performance monitoring**

💡 Advanced Features

1. Contextual Chunking

- **Configurable chunk size (250 words default)**
- **Overlap strategy (50 words) for context preservation**
- **Quality validation for extracted content**

2. Intelligent Query Handling

- **Pattern recognition for different query types**
- **Specialized extractors for requirements, citations, definitions**
- **Adaptive response formatting**

3. Source Attribution

- **Automatic source tracking per chunk**
- **Citation information in responses**
- **Document provenance maintenance**

4. Scalable Architecture

- **Modular agent design**
- **Extensible message protocol**
- **Async processing for performance**

🚀 RAG System - Challenges & Future Improvements

🔥 Challenges Faced During Development

1. Asynchronous State Management in Streamlit

Challenge: Streamlit's synchronous nature conflicted with async agent communication

- **Problem: Streamlit reruns the entire script on each interaction, making async state management complex**
- **Solution: Used st.session_state to persist message bus and agents across reruns**
- **Code Impact:**
- **python**
- `if "message_bus" not in st.session_state:`
- `    st.session_state.message_bus = MCPMessageBus()`
- `    # Initialize all agents in session state`

2. MCP Message Bus Synchronization

Challenge: Coordinating multiple agents with proper message delivery

- **Problem: Race conditions between message publishing and UI updates**
- **Solution: Implemented trace-based message tracking with active_traces dictionary**
- **Complexity: Had to handle message ordering and ensure UI updates trigger at right times**

3. Document Processing Memory Management

Challenge: Processing large documents without memory overflow

- **Problem: Large PDFs and multiple document uploads causing memory issues**
- **Solution:**
  - **Implemented chunking strategy with overlap**
  - **Used CPU-only embeddings to avoid GPU memory issues**
  - **Added content validation before processing**

4. FAISS Index Persistence

Challenge: Maintaining vector index across Streamlit reruns

- **Problem: FAISS index getting reinitialized on each app rerun**
- **Solution: Stored index and metadata in session state**
- **Trade-off: Memory usage increases with more documents**

5. Error Handling Across Agents

Challenge: Propagating errors through the agent chain

- **Problem: Errors in one agent breaking the entire workflow**
- **Solution: Implemented comprehensive error propagation with MCP error messages**
- **Example:**
- **python**

```
error_message = MCPMessage(
    sender="IngestionAgent",
    receiver="CoordinatorAgent",
    type="ERROR",
    trace_id=message.trace_id,
    payload={"error": f"Error processing {filename}: {str(e)}"}
)
```

6. LLM Integration Reliability

Challenge: Handling API failures and rate limits

- **Problem: Google Gemini API errors breaking the response flow**
- **Solution: Implemented fallback rule-based response system**
- **Backup Strategy: Pattern-based answer synthesis when LLM unavailable**

7. Real-time UI Updates

Challenge: Showing processing progress in real-time

- **Problem: Long document processing without user feedback**
- **Solution: Used st.spinner() and st.rerun() for dynamic updates**
- **User Experience: Added status indicators and progress feedback**

8. Text Extraction Quality

Challenge: Inconsistent text extraction from different file formats

- **Problem: PDFs with complex layouts, corrupted files, or empty content**
- **Solution: Added content validation and quality checks**
- **Fallback: Graceful degradation with informative error messages**

🔮 Future Scope & Improvements

1. Enhanced Document Processing

Advanced OCR Integration

- **Improvement: Add OCR for scanned PDFs and images**
- **Technology: Integrate Tesseract or cloud OCR services**
- **Impact: Support for non-text documents and better extraction**

Multi-modal Document Support

- **Improvement: Process images, tables, and charts within documents**
- **Technology: Vision transformers for image understanding**
- **Use Case: Research papers with figures and technical documentation**

Document Structure Preservation

- **Improvement: Maintain document hierarchy and formatting**
- **Technology: Document AI models for structure recognition**
- **Benefit: Better context understanding and navigation**

2. Advanced RAG Capabilities

Hybrid Search

- **Improvement: Combine semantic and keyword search**
- **Technology: BM25 + vector search fusion**
- **Benefit: Better recall for specific terms and concepts**

Multi-hop Reasoning

- **Improvement: Chain reasoning across multiple documents**
- **Technology: Graph-based knowledge representation**
- **Use Case: Complex queries requiring synthesis from multiple sources**

Temporal Awareness

- **Improvement: Track document versions and timestamps**
- **Technology: Temporal embeddings and versioning**
- **Application: Legal documents, policy changes, and updates**

3. Enhanced Agent Architecture

Specialized Agents

- **SummarizationAgent: Automatic document summarization**
- **ValidationAgent: Fact-checking and source verification**
- **PersonalizationAgent: User-specific response customization**
- **AnalyticsAgent: Usage patterns and performance monitoring**

Agent Load Balancing

- **Improvement: Distribute processing across multiple agent instances**
- **Technology: Message queue with worker pools**
- **Benefit: Better scalability and fault tolerance**

Smart Agent Routing

- **Improvement: Dynamic agent selection based on query type**

- **Technology: Query classification and routing logic**
- **Impact: Optimized processing paths for different use cases**

4. Advanced UI/UX Features

Interactive Document Viewer

- **Feature: In-app document viewing with highlighting**
- **Technology: PDF.js integration with annotation support**
- **Benefit: Visual context for retrieved information**

Conversation Memory

- **Feature: Multi-turn conversation with context retention**
- **Technology: Conversation state management**
- **Enhancement: Follow-up questions and clarifications**

Advanced Search Filters

- **Feature: Filter by document type, date, source, etc.**
- **Technology: Faceted search with metadata indexing**
- **User Experience: Refined search capabilities**

Collaborative Features

- **Feature: Shared document collections and annotations**
- **Technology: Real-time collaboration with WebSockets**
- **Use Case: Team knowledge sharing and research**

5. Performance & Scalability

Vector Database Migration

- **Improvement: Move from FAISS to production vector DB**
- **Technology: Pinecone, Weaviate, or Milvus**
- **Benefits: Better scalability, persistence, and multi-user support**

Caching Strategy

- **Improvement: Intelligent caching for embeddings and responses**
- **Technology: Redis or Memcached integration**
- **Impact: Faster response times and reduced API calls**

Batch Processing

- **Improvement: Process multiple documents simultaneously**
- **Technology: Async batch processing with progress tracking**
- **Scalability: Handle large document collections efficiently**

6. Security & Privacy

Document Access Control

- **Feature: User-based document permissions**
- **Technology: Role-based access control (RBAC)**
- **Security: Ensure data privacy and access restrictions**

Data Encryption

- **Improvement: Encrypt stored documents and embeddings**
- **Technology: AES encryption for data at rest**
- **Compliance: Meet enterprise security requirements**

Audit Logging

- **Feature: Comprehensive activity logging**
- **Technology: Structured logging with audit trails**
- **Purpose: Security monitoring and compliance**

7. Enterprise Features

API Integration

- **Feature: RESTful API for external integrations**
- **Technology: FastAPI with OpenAPI documentation**
- **Use Case: Integration with existing enterprise systems**

Deployment Options

- **Improvement: Container-based deployment**
- **Technology: Docker and Kubernetes support**
- **Benefit: Easy scaling and cloud deployment**

Configuration Management

- **Feature: Environment-specific configurations**
- **Technology: Config management with validation**
- **Flexibility: Adapt to different deployment scenarios**

8. Analytics & Monitoring

Usage Analytics

- **Feature: Track user interactions and popular queries**
- **Technology: Analytics dashboard with visualizations**
- **Insights: Understand user behavior and system performance**

Performance Monitoring

- **Improvement: Real-time performance metrics**
- **Technology: Prometheus and Grafana integration**

- **Metrics: Response times, error rates, and resource usage**

A/B Testing Framework

- **Feature: Test different response strategies**
- **Technology: Experiment framework with statistical analysis**
- **Optimization: Continuous improvement based on user feedback**

9. Advanced AI Features

Query Expansion

- **Improvement: Automatically expand queries with related terms**
- **Technology: Word embeddings and semantic expansion**
- **Benefit: Better retrieval coverage and accuracy**

Answer Validation

- **Feature: Fact-checking and confidence scoring**
- **Technology: Ensemble models and cross-validation**
- **Quality: Improved response reliability**

Personalized Recommendations

- **Feature: Suggest relevant documents and queries**
- **Technology: Collaborative filtering and user modeling**
- **Experience: Proactive information discovery**

10. Integration Ecosystem

Third-party Integrations

- **Slack/Teams: Direct integration with communication platforms**
- **Google Drive/SharePoint: Cloud storage connectivity**
- **Confluence/Notion: Knowledge base integration**
- **CRM Systems: Customer data integration**

? Query Processing Workflow

User Query
Chat Interface

→

CoordinatorAgent
Query Handler

→

RetrievalAgent
Context Retriever

→

LLMResponseAgent
Response Generator

RETRIEVE_CONTEXT
CoordinatorAgent → RetrievalAgent: Query with top_k parameter for chunk retrieval

CONTEXT_RESPONSE
RetrievalAgent → LLMResponseAgent: Ranked chunks with similarity scores

FINAL_RESPONSE
LLMResponseAgent → CoordinatorAgent: Generated response with source attribution

Model Context Protocol (MCP) Integration

Message Structure

Standardized communication with sender, receiver, type, trace_id, and payload fields

Publish-Subscribe

Agents subscribe to message bus for decoupled communication

Trace Tracking

End-to-end request tracing for debugging and monitoring

Async Processing

Non-blocking message handling for responsive UI

MCP Message Bus Architecture

CoordinatorAgent

IngestionAgent

RetrievalAgent

LLMResponseAgent

All agents communicate through the centralized message bus

All agents communicate through the centralized message bus

🛠️ Technology Stack

Core Framework

- Streamlit - Web UI
- Python 3.8 - Runtime
- asyncio - Async processing
- dataclasses - Message structure

Document Processing

- PyPDF2 - PDF extraction
- python-docx - Word processing
- python-pptx - PowerPoint
- pandas - CSV analysis

ML & Search

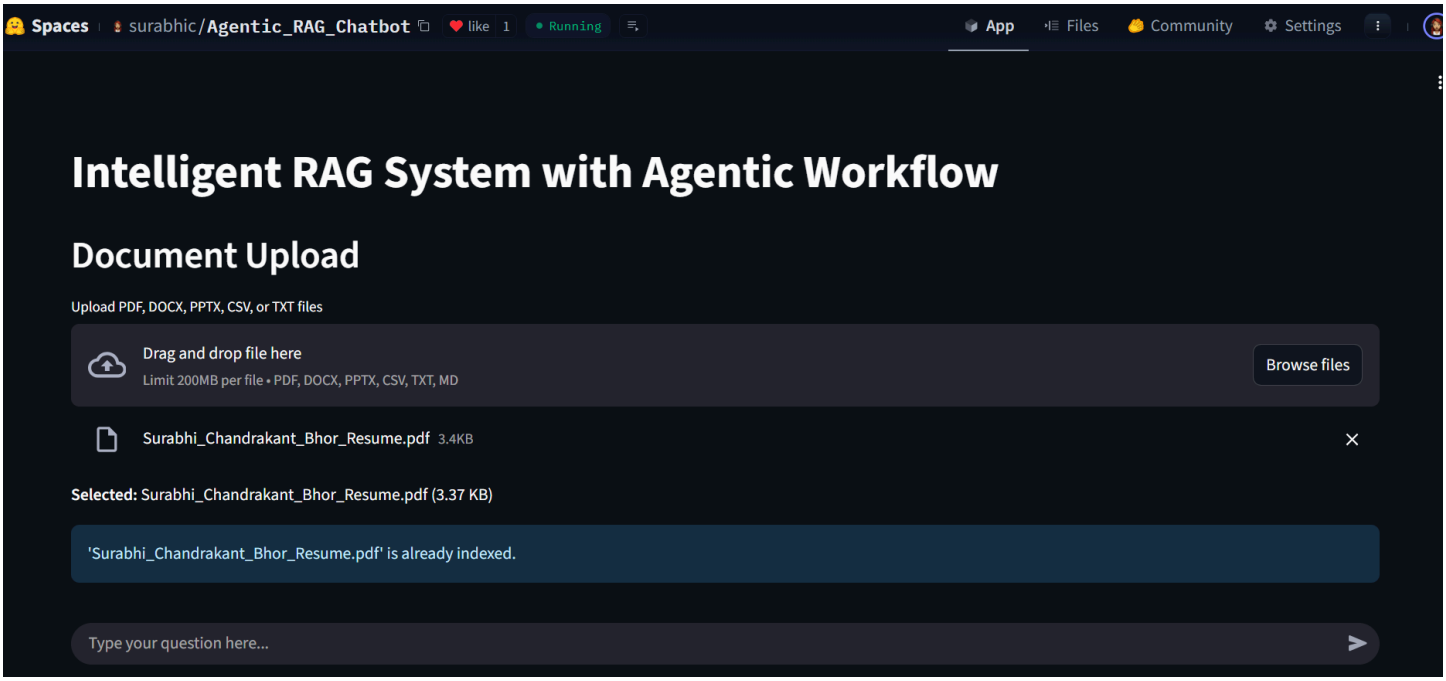
- SentenceTransformers - Embeddings
- FAISS - Vector search
- NumPy - Numerical ops
- all-MiniLM-L6-v2 - Model

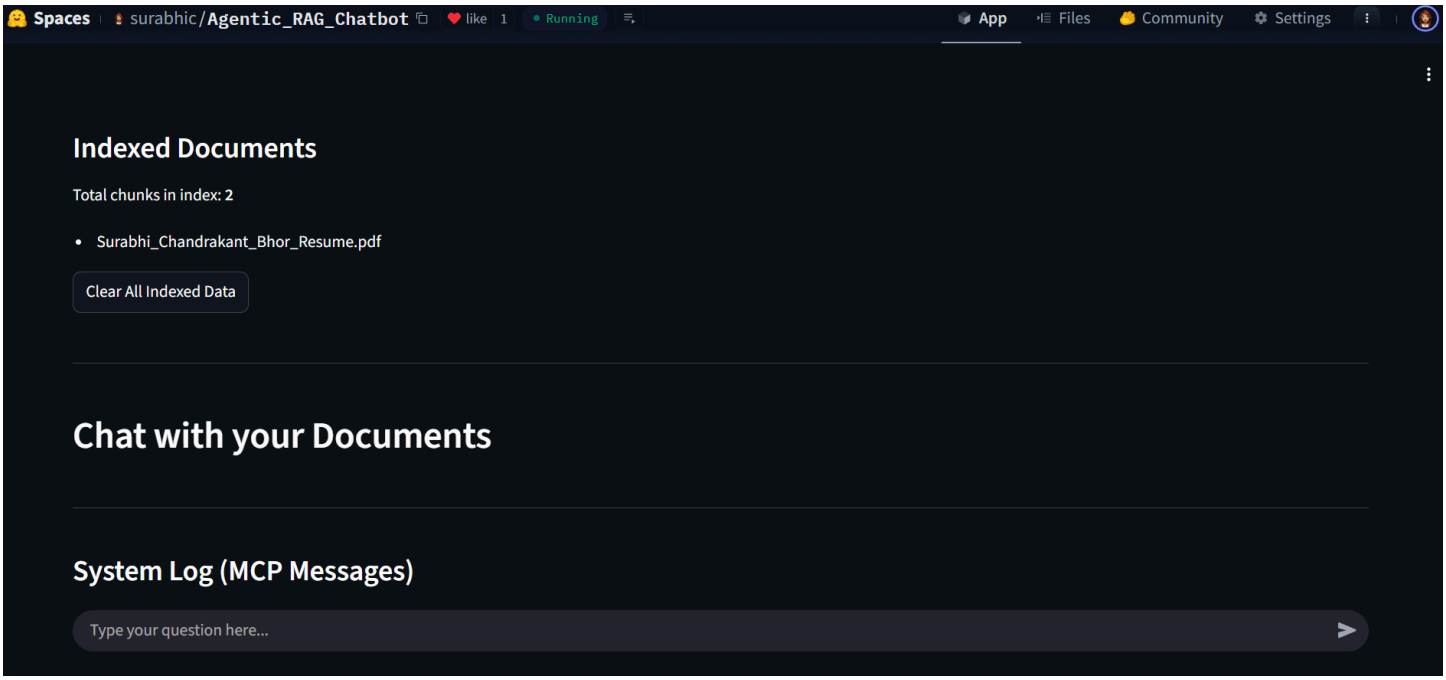
LLM Integration

- Google Generative AI - Gemini
- Custom fallback - Rule-based
- Context synthesis - RAG
- Source attribution - Citations

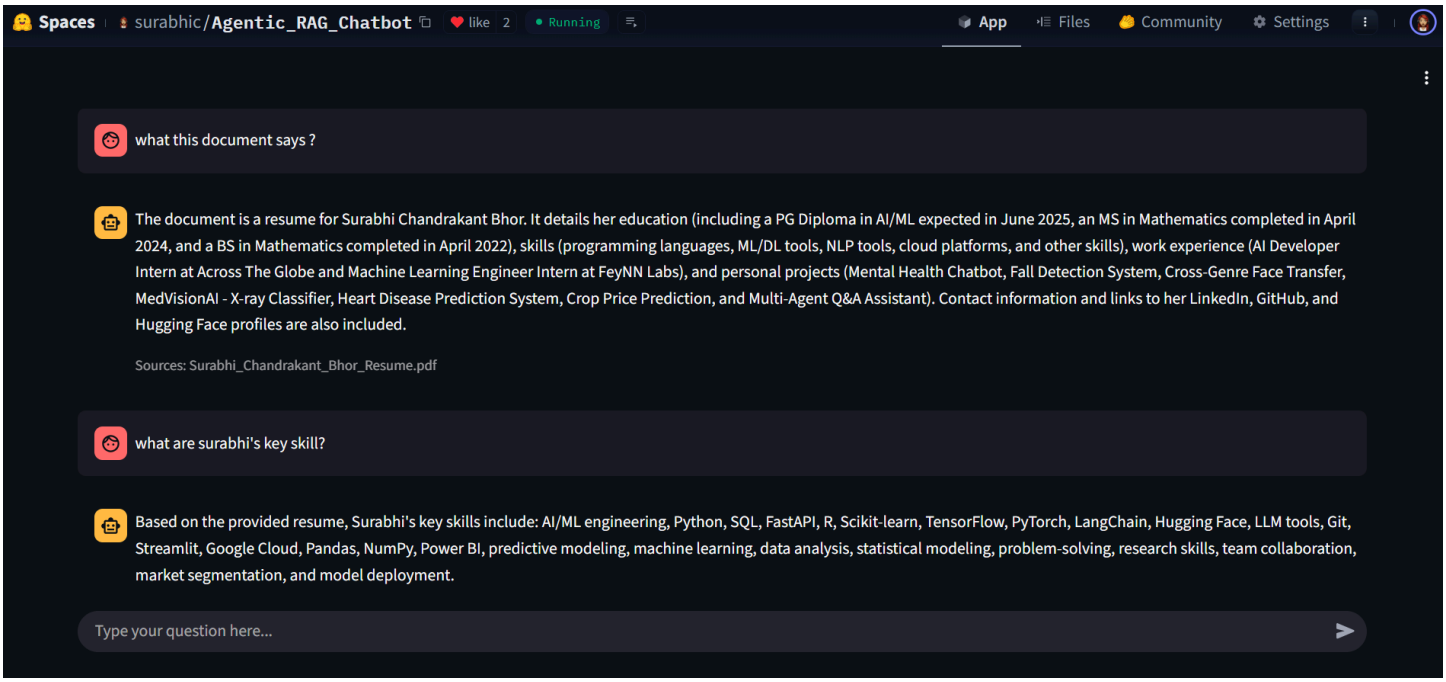
UI Screenshots of working app

Screenshot 1: Document Upload and Indexed Files





Screenshot 2: Chat Interface



Screenshot 3: System Log (MCP Messages)