Streamlit RAG Chatbot - Architecture Document

🏗️ Agent-Based Architecture with MCP Integration

Git-ayushkum1310/Slris-Project: Keep Husteling

Architecture Overview

The Streamlit RAG Chatbot is built using a multi-agent architecture pattern with Model Context Protocol (MCP) integration. This design promotes modularity, scalability, and maintainability by separating concerns into specialized agents.

Core Components

1. Message Bus (MCP Integration)

- **Purpose**: Central communication hub for all agents
- **Functionality**:
  - Handles message routing between agents
  - Maintains message logs for debugging and monitoring
  - Provides asynchronous communication patterns
  - Implements message tracing with unique trace IDs

2. Coordinator Agent

- **Role**: Master orchestrator of the system
- **Responsibilities**:
  - Receives user queries from the UI
  - Determines the appropriate workflow for each query
  - Coordinates between different agents
  - Manages the overall query processing pipeline

3. Ingestion Agent

- **Role**: Document processing and indexing
- **Responsibilities**:
  - Processes uploaded documents (PDF, PPTX, DOCX, CSV, TXT, MD)
  - Extracts text content from various file formats
  - Chunks documents into manageable pieces
  - Creates embeddings for semantic search

- Stores processed data in the knowledge base

4. Retrieval Agent

- **Role**: Information retrieval and context gathering
- **Responsibilities**:
  - Performs semantic search on the knowledge base
  - Retrieves relevant document chunks based on user queries
  - Ranks and filters search results
  - Provides context to the LLM Response Agent

5. LLM Response Agent

- **Role**: Natural language generation and response creation
- **Responsibilities**:
  - Integrates with Google's Gemini API
  - Processes retrieved context and user queries
  - Generates coherent, contextually relevant responses
  - Maintains conversation context and history

Agent Communication Pattern

```
UI → CoordinatorAgent → RetrievalAgent → LLMResponseAgent → UI
     ↓
     IngestionAgent (for document uploads)
```

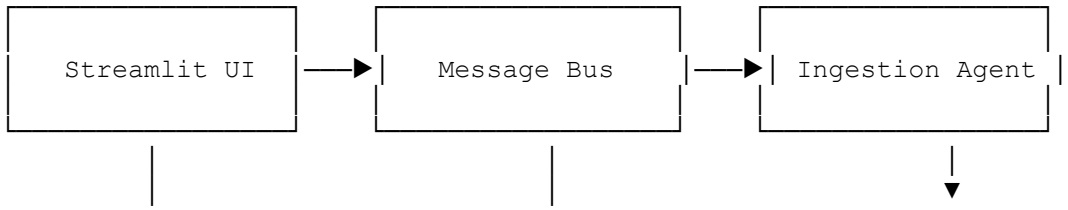
MCP Integration Benefits

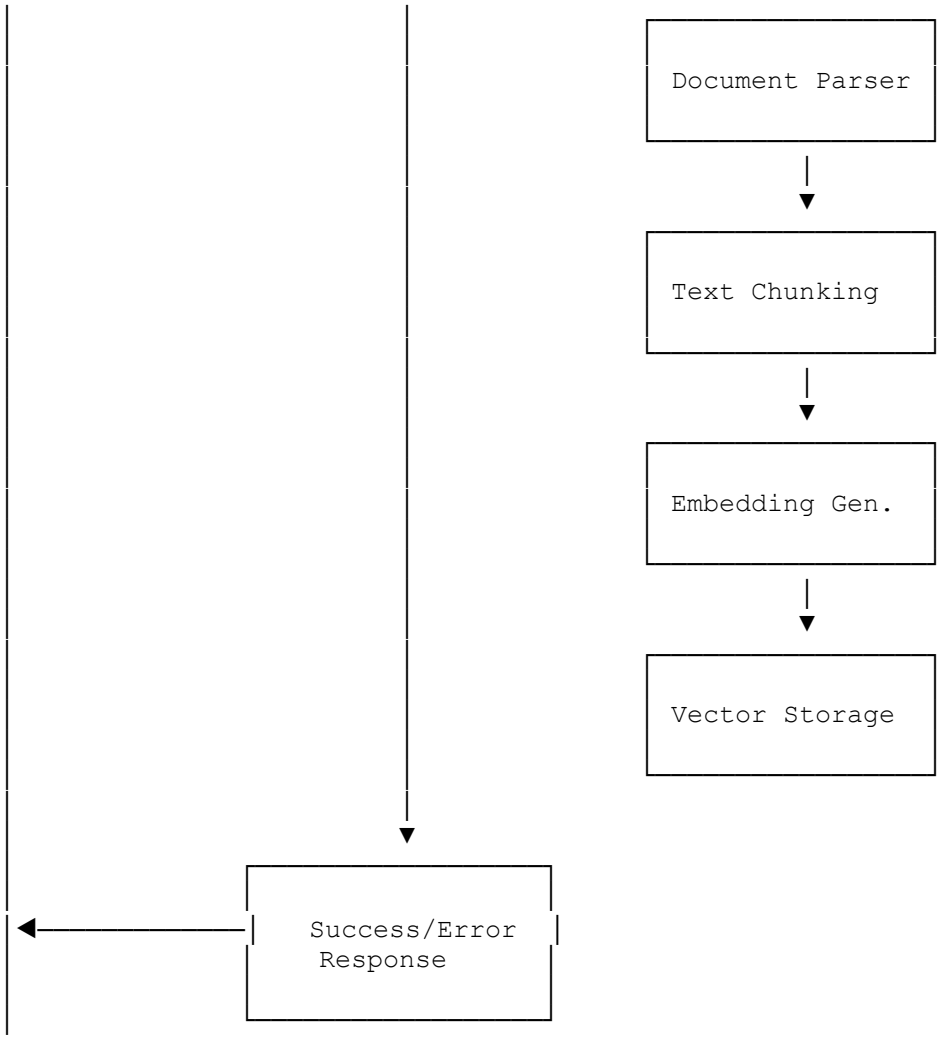
1. **Decoupled Architecture**: Each agent operates independently
2. **Scalable Communication**: Message bus handles complex routing
3. **Debugging & Monitoring**: Complete message trail for troubleshooting
4. **Fault Tolerance**: Agents can handle failures gracefully
5. **Extensibility**: Easy to add new agents or modify existing ones

---

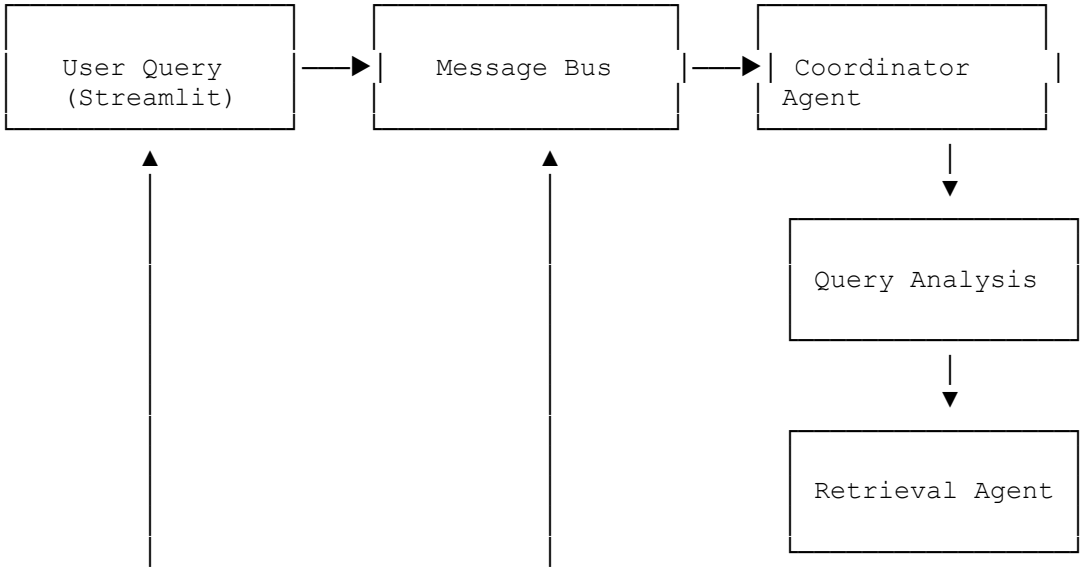
System Flow Diagram

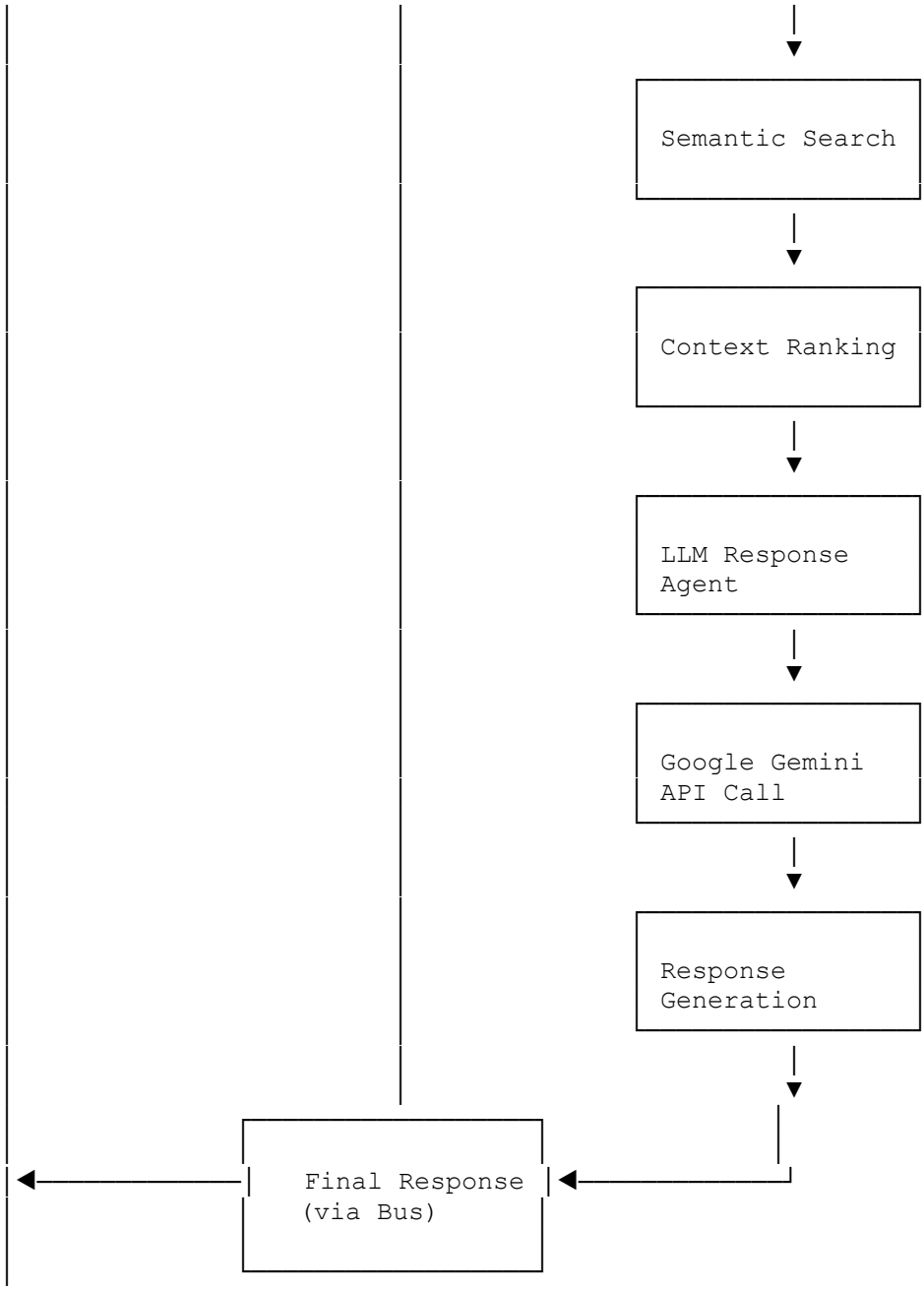
Document Upload Flow



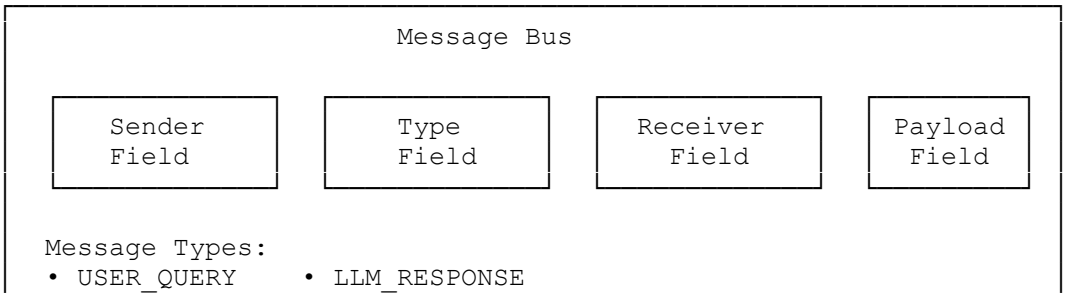


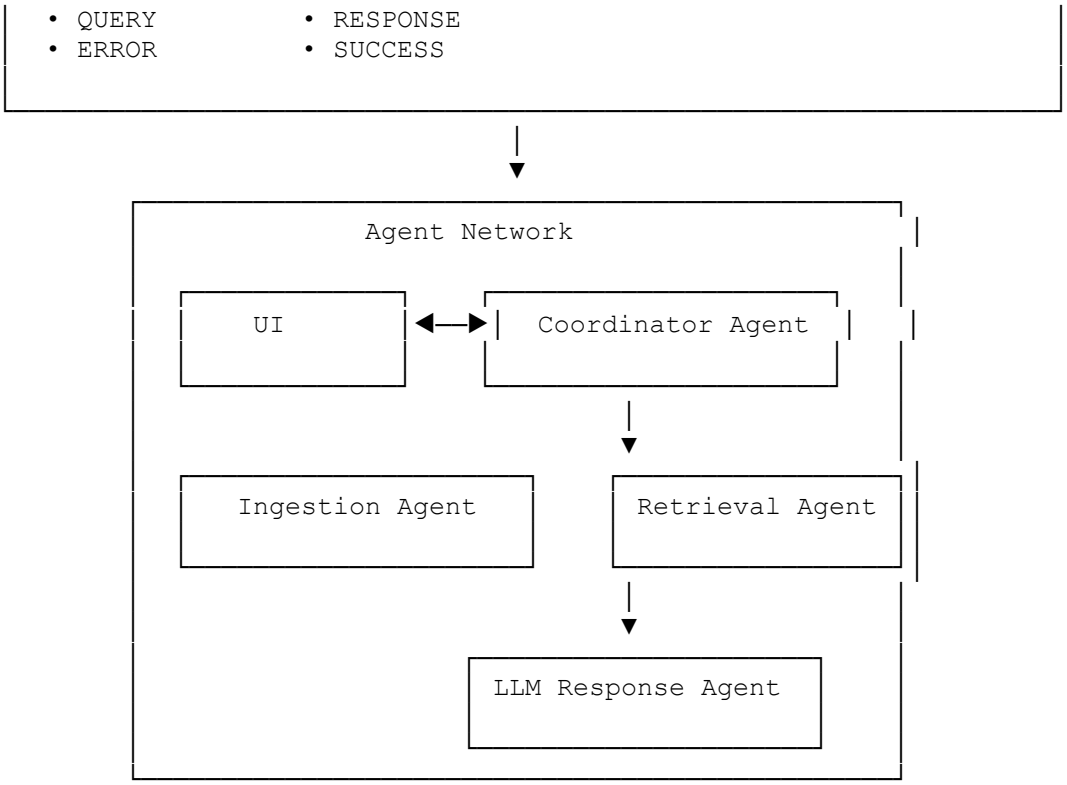
Query Processing Flow





Message Passing Architecture





🛠️ Tech Stack Used

Frontend & UI

- **Streamlit**: Main web application framework
  - Version: Latest stable
  - Components: chat_input, file_uploader, sidebar, tabs, metrics
  - Features: Session state management, real-time updates

Backend Architecture

- **Python**: Core programming language
  - Version: 3.8+
  - Async/await patterns for agent communication
  - Object-oriented design with inheritance

Agent Framework

- **Custom MCP Implementation**: Message bus and agent system
  - Message routing and logging
  - Agent lifecycle management
  - Trace ID tracking for debugging

AI/ML Components

- **Google Gemini API**: Large Language Model integration
  - Model: gemini-pro
  - API Key authentication
  - Response generation and context handling

Document Processing

- **File Format Support**:
  - PDF: PyPDF2 or pdfplumber
  - PPTX: python-pptx
  - DOCX: python-docx
  - CSV: pandas
  - TXT/MD: native Python file handling

Vector Search & Embeddings

- **Embedding Generation**:
  - Google's text-embedding models
  - OpenAI embeddings (alternative)
- **Vector Storage**:
  - In-memory storage for development
  - Extensible to external vector databases

Development Tools

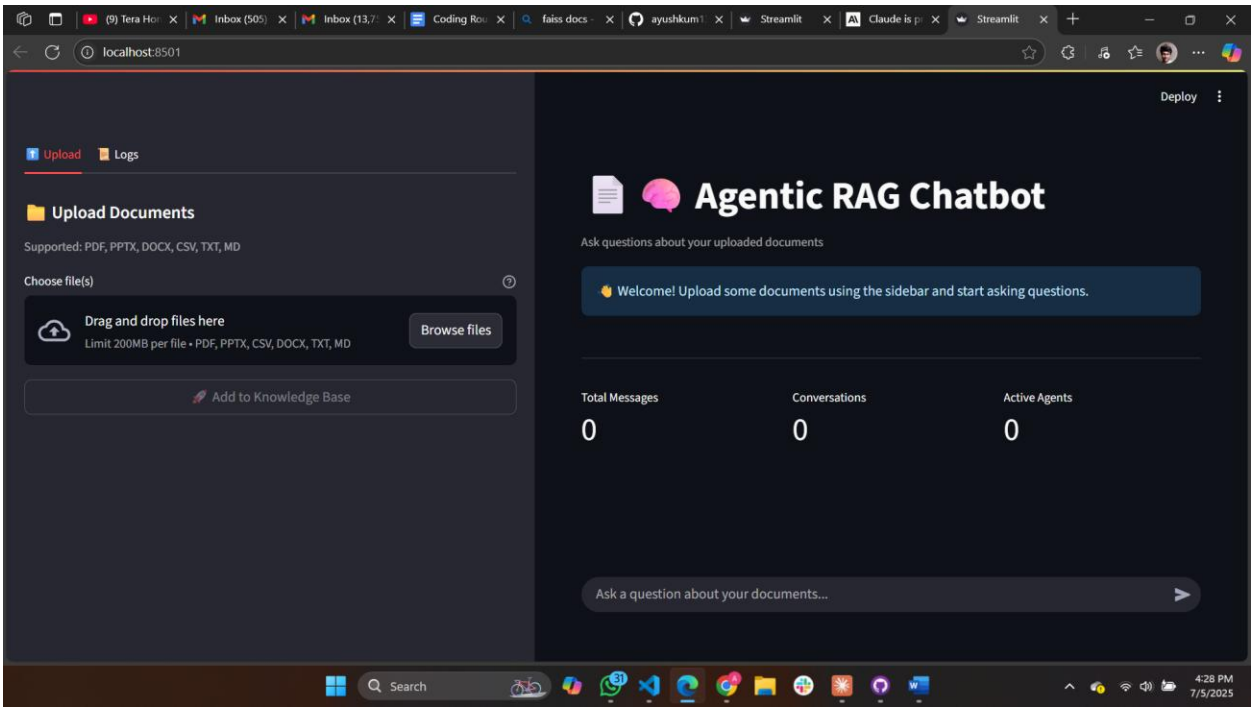
- **IDE**: VS Code, PyCharm
- **Version Control**: Git
- **Package Management**: pip/conda
- **Testing**: pytest (for future implementation)

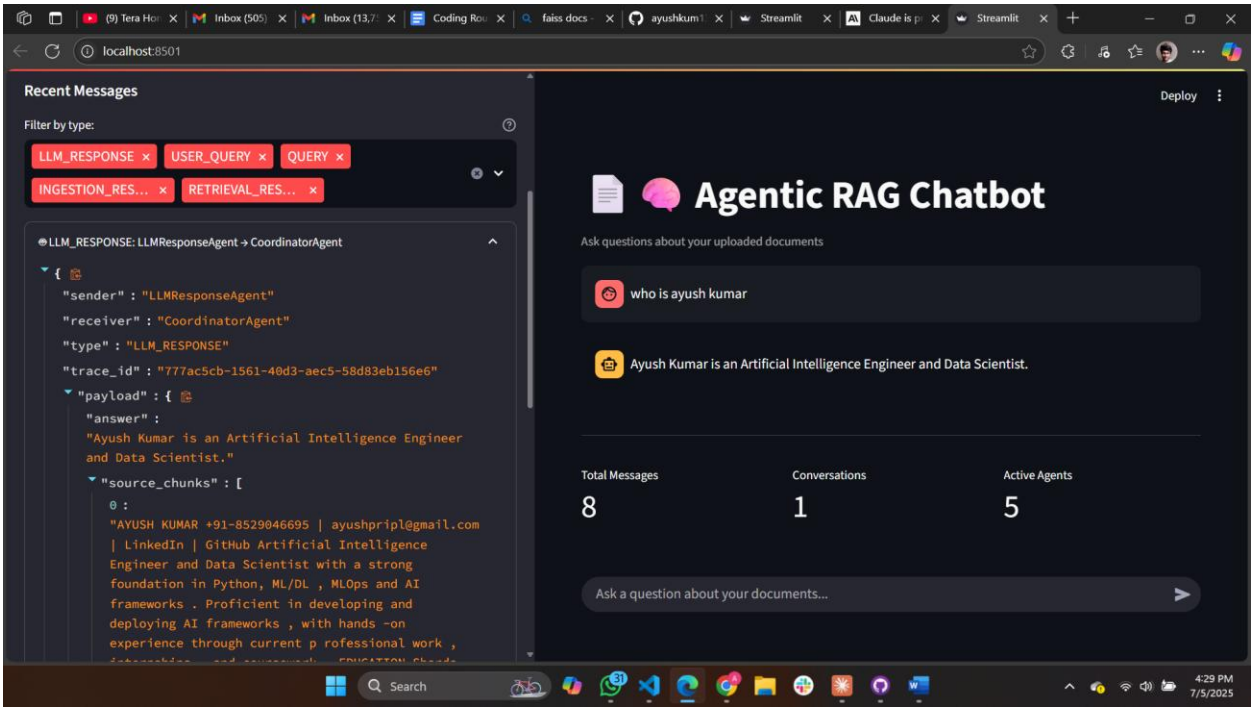
Deployment

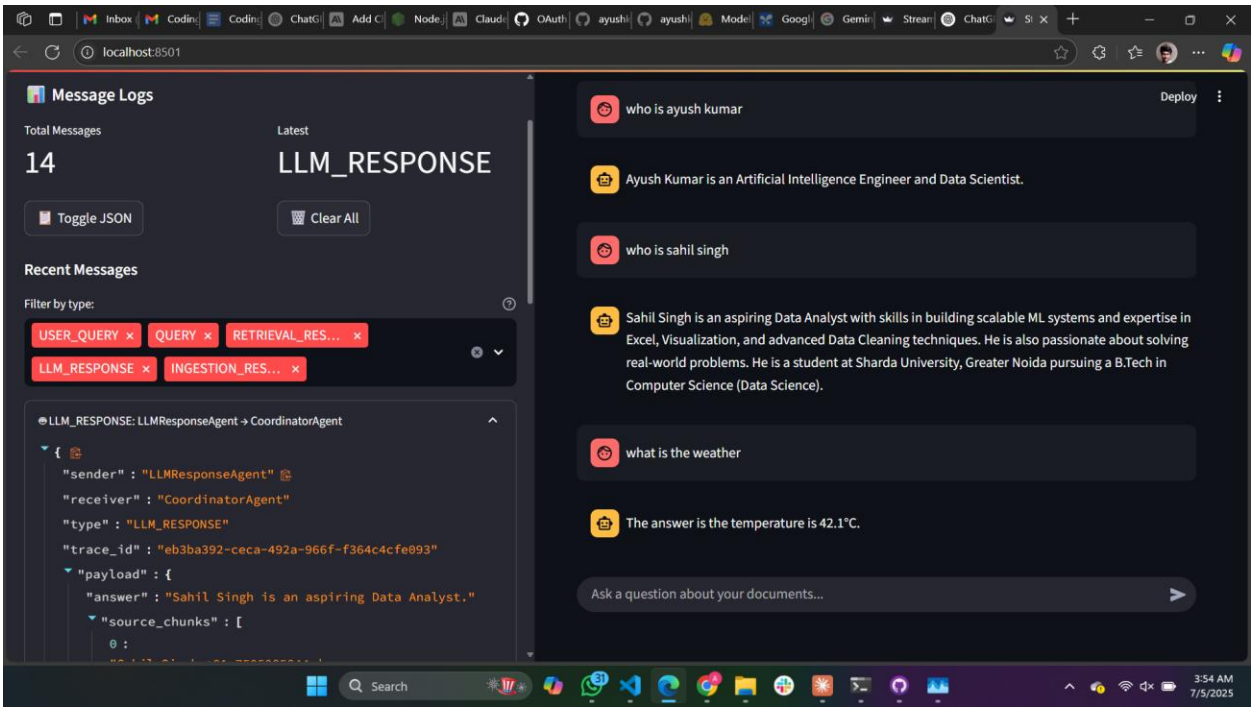
- **Local Development**: Direct Python execution
- **Production Options**:
  - Docker containerization
  - Cloud deployment (AWS, GCP, Azure)
  - Streamlit Cloud

📸 UI Screenshots

Main Chat Interface


localhost:8501
Deploy
Upload
Logs
Upload Documents
Supported: PDF, PPTX, DOCX, CSV, TXT, MD
Choose file(s)
Drag and drop files here
Limit 200MB per file • PDF, PPTX, CSV, DOCX, TXT, MD
Browse files
Add to Knowledge Base
Agentic RAG Chatbot
Ask questions about your uploaded documents
Welcome! Upload some documents using the sidebar and start asking questions.
Total Messages
0
Conversations
0
Active Agents
0
Ask a question about your documents...
Search
4:28 PM
7/5/2025


localhost:8501
Recent Messages
Filter by type:
LLM_RESPONSE
USER_QUERY
QUERY
INGESTION_RES...
RETRIEVAL_RES...
LLM_RESPONSE: LLMResponseAgent → CoordinatorAgent
"sender" : "LLMResponseAgent"
"receiver" : "CoordinatorAgent"
"type" : "LLM_RESPONSE"
"trace_id" : "777ac5cb-1561-40d3-aec5-58d83eb156e6"
"payload" : {
"answer" :
"Ayush Kumar is an Artificial Intelligence Engineer and Data Scientist."
"source_chunks" : [
0 :
"AYUSH KUMAR +91-8529046695 | ayushpripl@gmail.com | LinkedIn | GitHub Artificial Intelligence Engineer and Data Scientist with a strong foundation in Python, ML/DL , MLOps and AI frameworks . Proficient in developing and deploying AI frameworks , with hands -on experience through current p rofessional work ,
Deploy
Agentic RAG Chatbot
Ask questions about your uploaded documents
who is ayush kumar
Ayush Kumar is an Artificial Intelligence Engineer and Data Scientist.
Total Messages
8
Conversations
1
Active Agents
5
Ask a question about your documents...
Search
4:29 PM
7/5/2025



🚧 Challenges Faced During Development

1. Agent Communication Complexity

- **Challenge**: Designing a robust message passing system between agents
- **Solution**: Implemented centralized message bus with standardized message format
- **Learning**: Importance of clear communication protocols in distributed systems

2. Streamlit State Management

- **Challenge**: Managing complex application state across user sessions
- **Solution**: Utilized `st.session_state` effectively with proper state cleanup
- **Learning**: Streamlit's reactive nature requires careful state management

3. File Processing Reliability

- **Challenge**: Handling various file formats and edge cases
- **Solution**: Implemented robust error handling and fallback mechanisms
- **Learning**: Always plan for file format variations and corrupted files

4. Real-time UI Updates

- **Challenge**: Providing smooth user experience during long-running operations

- **Solution**: Implemented progress bars, status indicators, and async processing
- **Learning**: User feedback is crucial for perceived performance

5. Memory Management

- **Challenge**: Handling large documents and maintaining performance
- **Solution**: Implemented document chunking and efficient text processing
- **Learning**: Scalability considerations must be built-in from the start

6. LLM Integration

- **Challenge**: Managing API rate limits and response consistency
- **Solution**: Implemented retry logic and response validation
- **Learning**: Always have fallback strategies for external API dependencies

7. Debugging Multi-Agent Systems

- **Challenge**: Tracing issues across multiple agents
- **Solution**: Comprehensive logging with trace IDs and message history
- **Learning**: Observability is crucial in distributed systems

---

🔮 Future Scope & Improvements

Technical Enhancements

1. Advanced Vector Search

- **Current**: In-memory vector storage
- **Future**: Integration with specialized vector databases
  - Pinecone, Weaviate, or Chroma DB
  - Improved search accuracy and performance
  - Support for hybrid search (semantic + keyword)

2. Multi-Modal Document Support

- **Current**: Text-based documents only
- **Future**: Image, audio, and video processing
  - OCR for scanned documents
  - Speech-to-text for audio files
  - Video content analysis

3. Advanced Agent Capabilities

- **Current**: Basic agent coordination
- **Future**: Intelligent agent behaviors
    - Self-healing agents
    - Dynamic agent spawning based on workload
    - Agent performance monitoring and optimization

User Experience Improvements

4. Enhanced Chat Interface

- **Current**: Basic chat with text responses
- **Future**: Rich multimedia responses
    - Charts and graphs generation
    - Interactive visualizations
    - Citation and source linking

5. Collaboration Features

- **Current**: Single-user sessions
- **Future**: Multi-user collaboration
    - Shared knowledge bases
    - Real-time collaborative queries
    - User permissions and access control

6. Advanced Analytics

- **Current**: Basic message logging
- **Future**: Comprehensive analytics dashboard
    - Query performance metrics
    - User behavior analysis
    - System health monitoring

Scalability & Performance

7. Cloud-Native Architecture

- **Current**: Local deployment
- **Future**: Cloud-native design
    - Microservices architecture
    - Auto-scaling capabilities
    - Multi-region deployment

8. Advanced Caching

- **Current**: Session-based storage
- **Future**: Intelligent caching layers

    - Redis integration
    - Query result caching
    - Predictive pre-loading

AI/ML Improvements

9. Model Flexibility

- **Current**: Single LLM integration (Gemini)
- **Future**: Multi-model support
    - Model routing based on query type
    - Ensemble responses
    - Custom fine-tuned models

10. Intelligent Document Processing

- **Current**: Basic text extraction
- **Future**: Advanced understanding
    - Document structure recognition
    - Relationship extraction
    - Automatic summarization

Enterprise Features

11. Security & Compliance

- **Current**: Basic security
- **Future**: Enterprise-grade security
    - End-to-end encryption
    - Audit logging
    - GDPR compliance

12. Integration Capabilities

- **Current**: Standalone application
- **Future**: Enterprise integrations
    - API endpoints for external systems
    - Webhook support
    - Third-party service integrations

Development & Operations

13. DevOps & Monitoring

- **Current**: Manual deployment
- **Future**: Automated CI/CD

    - Automated testing suite
    - Performance monitoring
    - Error tracking and alerting

14. Configuration Management

- **Current**: Hardcoded configurations
- **Future**: Dynamic configuration
    - Environment-based configs
    - Runtime configuration updates
    - Feature flags

📋 Conclusion

The Streamlit RAG Chatbot represents a successful implementation of modern AI architecture principles, combining the power of large language models with intelligent document processing. The agent-based design ensures scalability and maintainability, while the MCP integration provides robust communication patterns.

The system demonstrates the potential of combining traditional software engineering practices with cutting-edge AI capabilities, creating a foundation for more sophisticated knowledge management systems.

Key Achievements

- ✅ Modular, extensible architecture
- ✅ Smooth user experience with real-time feedback
- ✅ Robust error handling and logging
- ✅ Multi-format document support
- ✅ Scalable agent communication system

Next Steps

1. Implement vector database integration
2. Add comprehensive testing suite
3. Optimize for production deployment
4. Enhance security and compliance features
5. Develop advanced analytics capabilities

This document serves as a comprehensive guide to the Streamlit RAG Chatbot architecture and provides a roadmap for future development.