Document Question-Answering Chatbot using Agentic RAG

Overview: Agent-Based Architecture with MCP

Our chatbot leverages a modular, agent-based architecture, seamlessly integrated with the Model Context Protocol (MCP) for robust communication.

Ingestion Agent

Parses diverse document formats (PDF, PPTX, DOCX, CSV, TXT, MD) ensuring comprehensive data extraction.

Retrieval Agent

Indexes and retrieves the most relevant text chunks using advanced vector search capabilities.

LLM Response Agent

Generates natural language answers by leveraging the provided context and user queries.

MCP Integration

Facilitates seamless, structured communication between all agents using a defined message protocol.

System Flow Diagram: Message Passing with MCP

The system orchestrates a precise flow of information, with each agent communicating via the Model Context Protocol (MCP) for efficient data exchange.

User Uploads Files

Initiates the process by uploading documents in various formats.

Ingestion Agent Extracts Text

Parses documents and returns the full content to the next stage.

Retrieval Agent Indexes

Indexes the extracted text and identifies top N relevant chunks.

LLM Response Agent Generates

Receives query and chunks (via MCP) to formulate natural language responses.

Response to User

Delivers the generated answer to the user via the Streamlit interface.

Core Technologies and Protocol Layer

Our solution leverages a robust tech stack, with a custom Model Context Protocol (MCP) as the backbone for inter-agent communication.

Frontend (Streamlit)

- Interactive user interface for seamless interaction.
- Intuitive design for document uploads and chatbot responses.

Protocol Layer (Custom MCP)

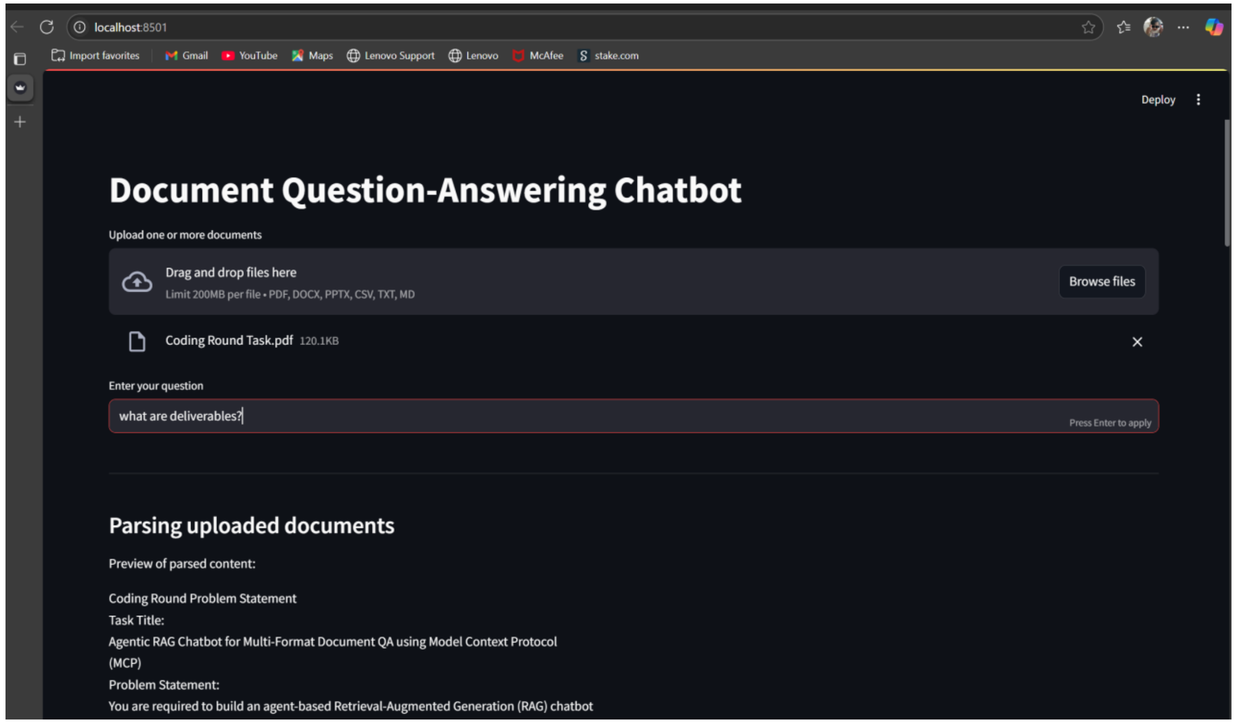
- Ensures standardised and traceable agent communication.
- Enables modularity and scalability across the system.

Backend Agents (Python Ecosystem)

- **Python:** Core development language.
- **FAISS:** Efficient vector similarity search.
- **Hugging Face Transformers:** For embeddings and LLM operations.
- **Tempfile & uuid:** For secure file handling and traceable messages.

User Interface Screenshots

Experience the intuitive design of our chatbot, from seamless document uploads to precise content previews and retrieved chunk displays.



Document Question-Answering Chatbot

Upload one or more documents

Drag and drop files here
Limit 200MB per file • PDF, DOCX, PPTX, CSV, TXT, MD

Coding Round Task.pdf 120.1KB

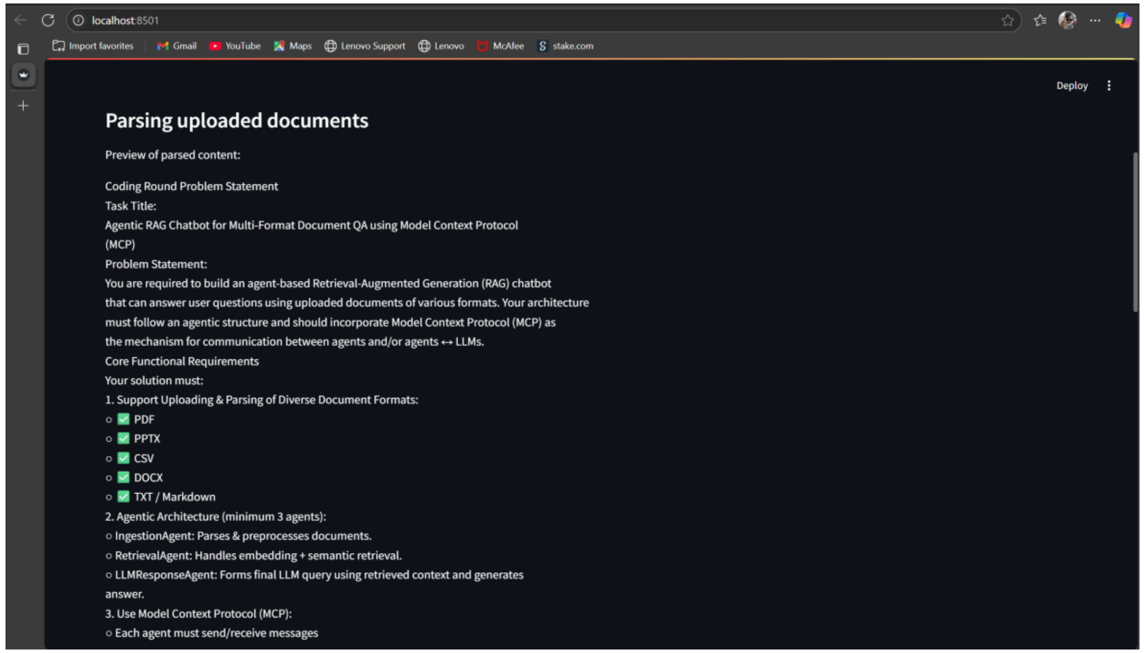
Enter your question

what are deliverables?

Parsing uploaded documents

Preview of parsed content:

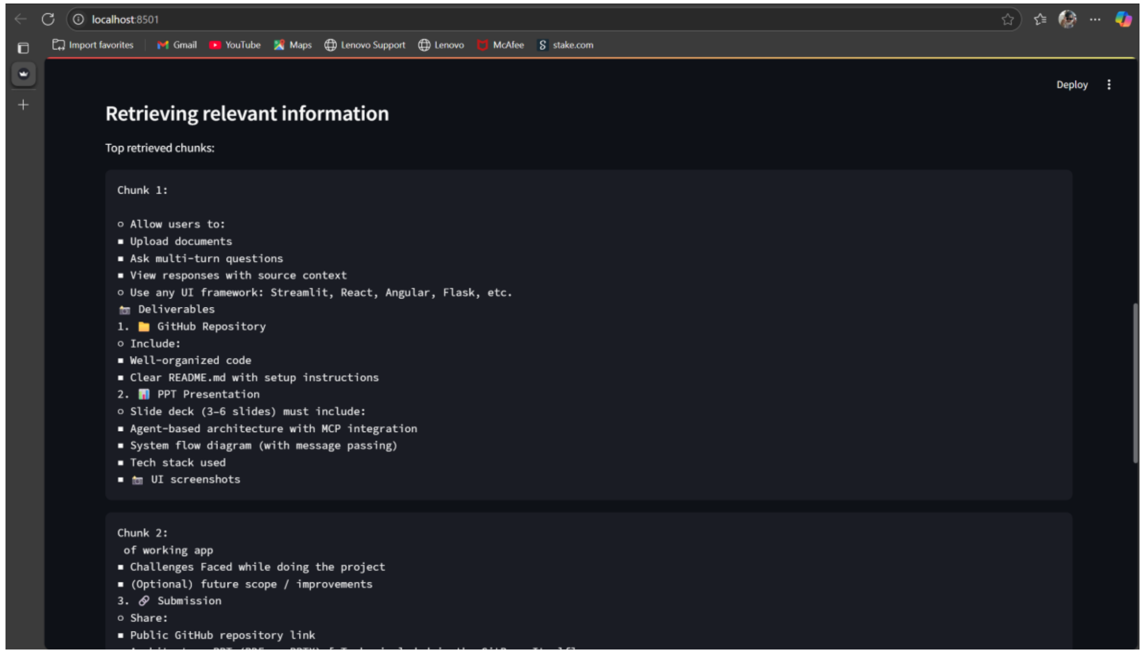
Coding Round Problem Statement
Task Title:
Agentic RAG Chatbot for Multi-Format Document QA using Model Context Protocol
(MCP)
Problem Statement:
You are required to build an agent-based Retrieval-Augmented Generation (RAG) chatbot



Parsing uploaded documents

Preview of parsed content:

Coding Round Problem Statement
Task Title:
Agentic RAG Chatbot for Multi-Format Document QA using Model Context Protocol
(MCP)
Problem Statement:
You are required to build an agent-based Retrieval-Augmented Generation (RAG) chatbot
that can answer user questions using uploaded documents of various formats. Your architecture
must follow an agentic structure and should incorporate Model Context Protocol (MCP) as
the mechanism for communication between agents and/or agents ↔ LLMs.
Core Functional Requirements
Your solution must:
1. Support Uploading & Parsing of Diverse Document Formats:
○ ✅ PDF
○ ✅ PPTX
○ ✅ CSV
○ ✅ DOCX
○ ✅ TXT / Markdown
2. Agentic Architecture (minimum 3 agents):
○ IngestionAgent: Parses & preprocesses documents.
○ RetrievalAgent: Handles embedding + semantic retrieval.
○ LLMResponseAgent: Forms final LLM query using retrieved context and generates
answer.
3. Use Model Context Protocol (MCP):
○ Each agent must send/receive messages



Retrieving relevant information

Top retrieved chunks:

```
Chunk 1:

○ Allow users to:
■ Upload documents
■ Ask multi-turn questions
■ View responses with source context
○ Use any UI framework: Streamlit, React, Angular, Flask, etc.
📷 Deliverables
1. 📁 GitHub Repository
○ Include:
■ Well-organized code
■ Clear README.md with setup instructions
2. 📊 PPT Presentation
○ Slide deck (3–6 slides) must include:
■ Agent-based architecture with MCP integration
■ System flow diagram (with message passing)
■ Tech stack used
■ 📷 UI screenshots
```

```
Chunk 2:
 of working app
■ Challenges Faced while doing the project
■ (Optional) future scope / improvements
3. 🔗 Submission
○ Share:
■ Public GitHub repository link
```

Key Challenges and Solutions

We addressed several technical hurdles to ensure the chatbot's performance and reliability, from complex parsing to UI optimisation.

Cross-Format Parsing

Custom logic developed for precise extraction from PPTX, DOCX, CSV, and other varied file types.

Agent Communication

Designed a generic and reusable Model Context Protocol (MCP) for clean message passing.

Chunking Logic Optimisation

Fine-tuned chunk sizes for optimal context relevance and efficient LLM token management.

Streamlit UX Limitations

Implemented strategies to manage large text volumes, preventing UI lag and crashes.

LLM Hallucination Mitigation

Addressed occasional irrelevant answers by refining chunking logic and context provision.

Key Takeaways and Next Steps

The Agentic RAG Chatbot powered by MCP offers a robust solution for document Q&A, with a clear roadmap for continued innovation.

1. **Modular Design**

   Agent-based architecture ensures scalability and maintainability for future growth.

2. **Seamless Communication**

   MCP guarantees reliable and structured data exchange between all components.

3. **Enhanced Accuracy**

   Refined chunking and context management minimise hallucinations for precise answers.