



AGENTIC RAG CHATBOT FOR MULTI- FORMAT DOCUMENT QA USING MODEL CONTEXT PROTOCOL (MCP)



By Karthik



TABLE OF CONTENTS

- Project Overview & Technology Stack
- Agentic Architecture with MCP Protocol
- System Flow & LangGraph Execution
- Gradio UI – Application Snapshots
- Challenges Faced During Development
- Future Scope & Enhancements



PROJECT OVERVIEW & TECHNOLOGY STACK

Design a smart and modular chatbot system that can extract answers from documents of various formats (PDF, PPTX, DOCX, CSV, TXT) using a Retrieval-Augmented Generation (RAG) architecture backed by agents.

🧠 Key Highlights:

- Agent-based design for modularity and scalability.
- Integrated LangGraph to simulate agent collaboration via Model Context Protocol (MCP).
- Multi-turn, real-time conversational UI using Gradio.
- Lightweight, fast, and deployed fully on Google Colab.

🔧 Tech Stack:

- LLM: Gemini 2.0 Flash (Fast + Accurate)
- Embeddings: Google Embeddings 001
- Frameworks: LangChain + LangGraph
- Vector Store: ChromaDB
- UI: Gradio
- Platform: Pycharm (runtime environment)

AGENTIC ARCHITECTURE WITH MCP PROTOCOL

Core Agents:

- IngestionAgent: Reads, parses, and chunkifies uploaded documents.
- RetrievalAgent: Computes embeddings and retrieves top matching chunks.
- LLMResponseAgent: Prepares final prompt using context and invokes the LLM.

Model Context Protocol (MCP):

- Custom JSON-like message format used for agent communication.
- Example Message:

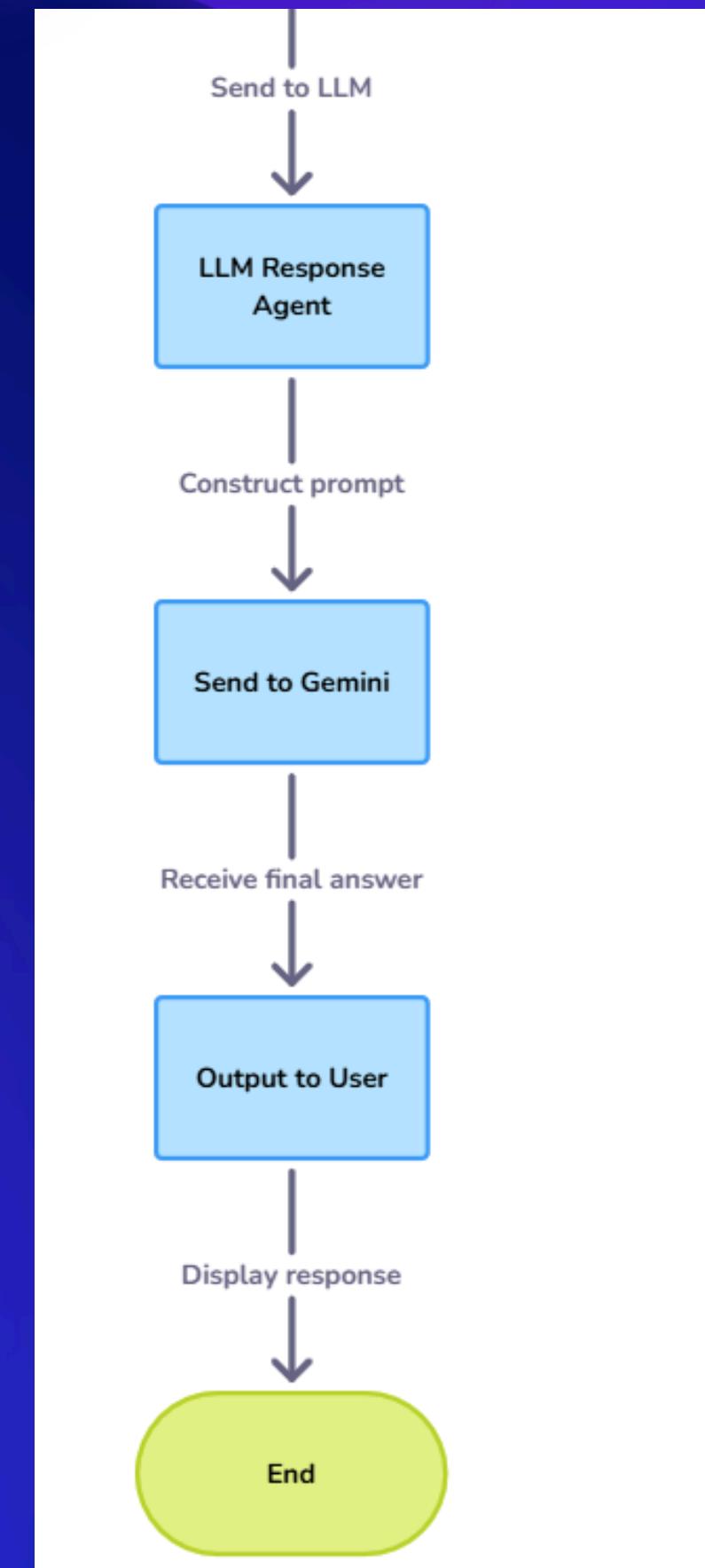
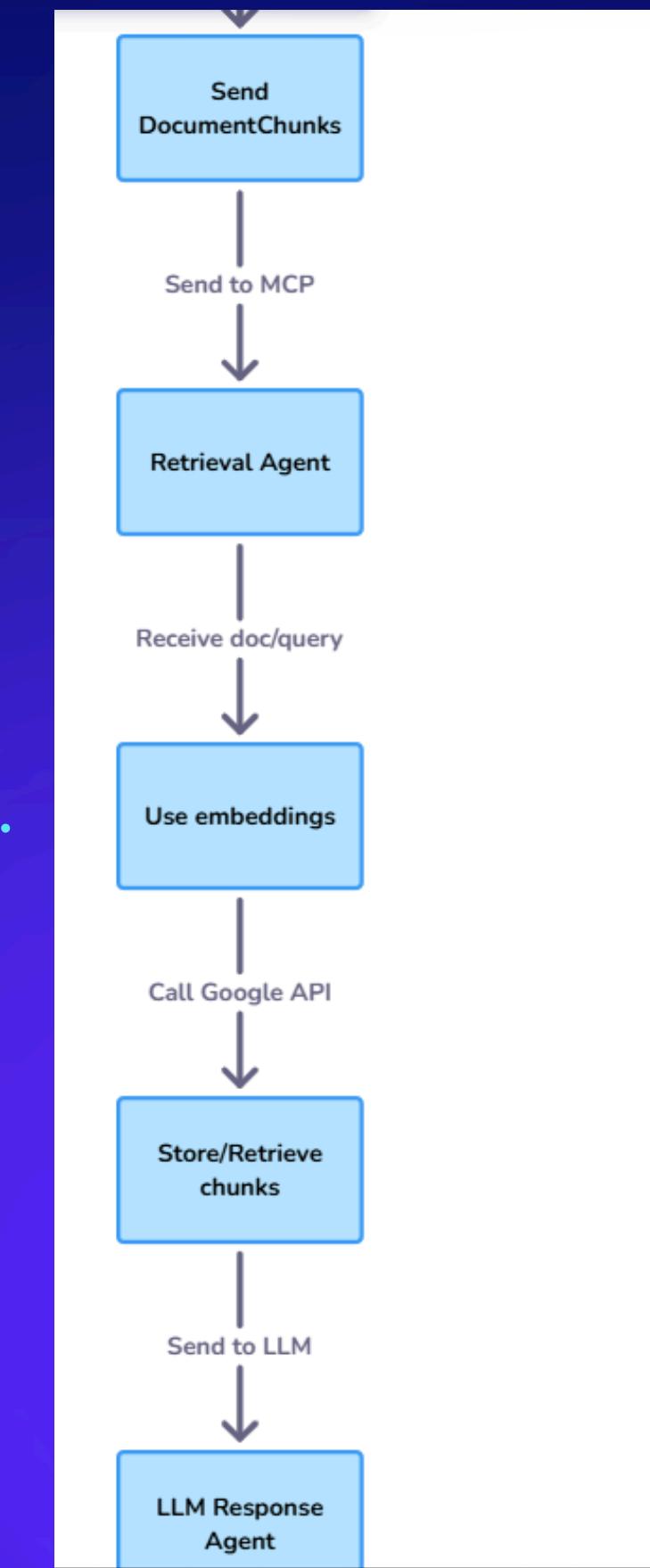
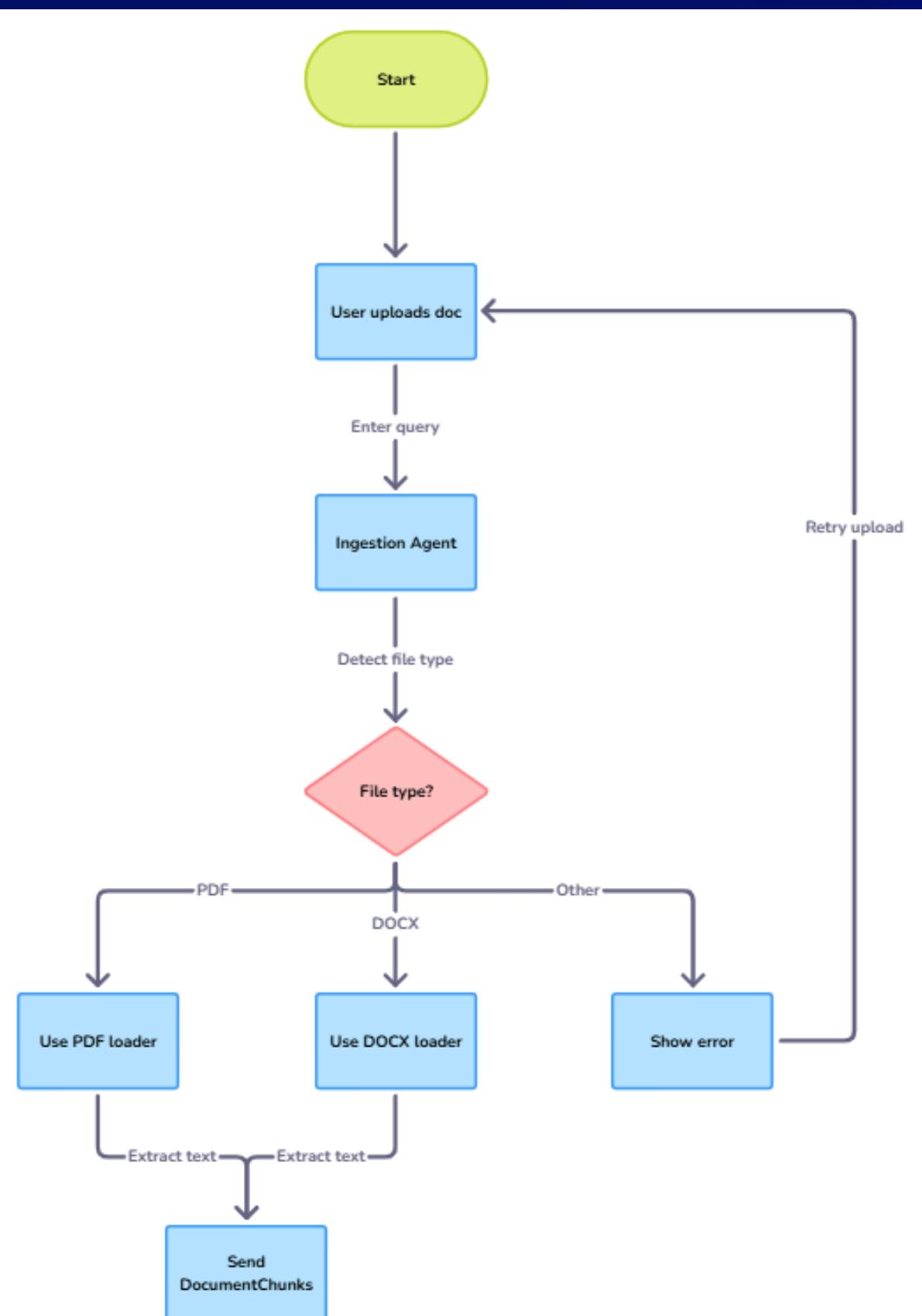
```
{  
    "type": "RETRIEVAL_RESULT",  
    "sender": "RetrievalAgent",  
    "receiver": "LLMResponseAgent",  
    "trace_id": "rag-457",  
    "payload": {  
        "retrieved_context": ["slide 3: revenue up", "doc: Q1  
summary..."],  
        "query": "What KPIs were tracked in Q1?"  
    }  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

Benefits:

Enables stateless, traceable, and scalable agent communication.

Decouples responsibilities for each task.

SYSTEM FLOW & LANGGRAPH EXECUTION



GRADIO UI - APPLICATION SNAPSHOTS

This screenshot shows the initial state of the "Agentic RAG Chatbot for Multi-Format Document" application. On the left, there is a large input area labeled "Upload Documents" with a placeholder "Drop File Here" and an "Upload" button below it. In the center, there is a text input field with the placeholder "Enter your question here" and a sub-placeholder "Ask something from your documents". At the bottom, there is a "Get Answer" button.

This screenshot shows the application after a file has been uploaded. The "Upload Documents" area now displays "GenAllInterview_questions.pdf" with a size of "71.2 KB". The central text input field still contains the placeholder text.

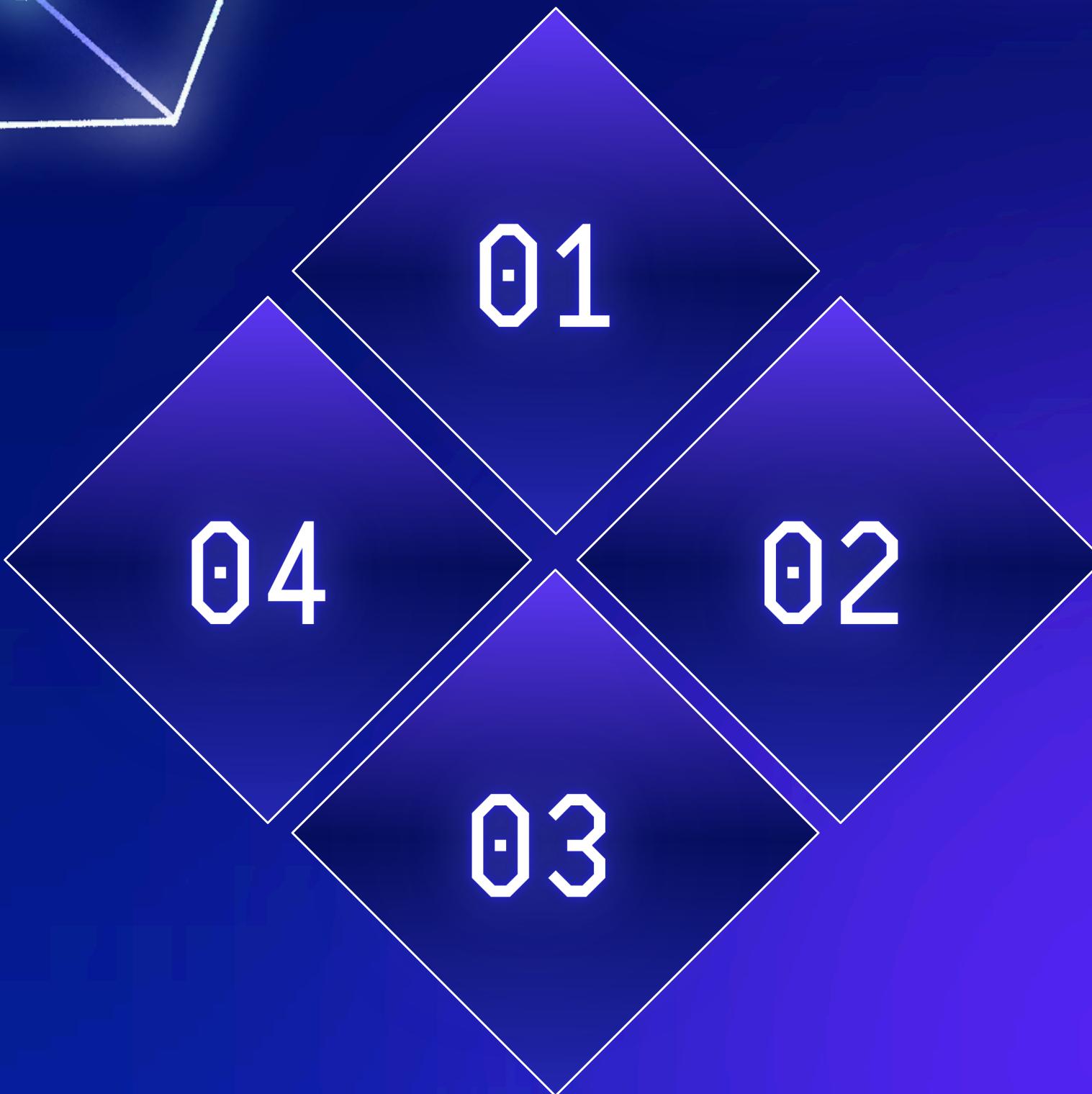
This screenshot shows the application after a question has been entered into the text input field. The question "What is Tokenization?" is visible. Below the input field, there is a "Get Answer" button. The "Answer" section at the bottom is currently empty.

This screenshot shows the application after the "Get Answer" button has been clicked. The "Answer" section now contains the following text: "Tokenization involves breaking down text into smaller units, or tokens, such as words, subwords, or characters. For example, the word \"artificial\" might be split into \"art,\" \"ifc,\" and \"ial.\" This process is vital for LLMs because they process numerical representations of tokens, not raw text. Tokenization enables models to handle diverse languages, manage rare or unknown words, and optimize vocabulary size, enhancing computational efficiency and model performance."

CHALLENGES FACED DURING DEVELOPMENT

- 🧠 Initial LLM integration issues:
- Originally explored VertexAI but replaced it with Gemini Flash due to API restrictions on Colab.
- 📁 Multi-format parsing difficulties:
- Handling edge cases in malformed or image-heavy PDFs and inconsistent DOCX formatting.
- 🔄 LangGraph Complexity:
- Understanding node-based agent flow and structuring MCP message passing took time.
- Initial struggles with Gradio deployment on Colab due to ngrok connection issues so changed to pycharm.

FUTURE SCOPE & ENHANCEMENTS



- More File Support: Add OCR for scanned documents and images (e.g., invoice PDFs).
- User Authentication: Allow per-user history, document access control.
- Conversation Memory: Use LangChain memory components for multi-turn coherent chat.
- Response Confidence Scores: Let users understand the confidence of answers.
- Experiment with Local LLMs: Reduce cloud dependence and improve privacy.

THANK YOU!

