Capstone Project: PDF Ingestion and NLI for Data Querying (Multi-Agent Setup)

---

1. Project Overview

This capstone focuses on building a multi-agent generative AI system capable of ingesting, understanding, and querying data from complex PDF documents. The system should process various data types within the PDFs (text, tables, charts, and images) and allow users to interact with it through a natural language interface (NLI). It must extract meaningful insights and provide responses that are contextually relevant, accurate, and human-readable.

2. Objective

Develop a **multi-agent AI pipeline** that can automatically parse PDFs and provide natural language responses to user queries based on extracted information.

3. Scenario-Based Problem Statement

A multinational enterprise receives thousands of quarterly financial reports from subsidiaries across the globe in PDF format. Each document contains structured (tables, charts) and unstructured (narrative text) data. Manually extracting KPIs such as revenue trends, cost breakdowns, or risk commentary is time-consuming and error-prone.

To address this, the company aims to implement an intelligent AI system that: - Ingests all PDF reports into a centralized repository. - Parses and extracts all relevant data elements (text, numbers, tables, charts). - Allows users to interact with the ingested content through a **Natural Language Interface (NLI)**. - Uses specialized agents to manage data extraction, context understanding, and response generation.

Example queries: - "Summarize the revenue performance for 2022 vs 2023." - "Identify the top three expense categories in the Q4 report." - "What is the trend in operational efficiency over the last five years?"

4. Expected Outcomes

- **PDF Ingestion Pipeline** capable of handling mixed content.
- **Multi-Agent Architecture** for modular task handling.
- **NLI Query System** for interactive data retrieval.
- **Domain-Aware Summarization and Reporting** features.
- **Integration Layer** for connecting AI agents and visualization tools.

5. System Architecture

Key Components: 1. **Ingestion Agent** – Extracts text, tables, and metadata from PDF using tools like `PyMuPDF`, `PDFMiner`, or `Camelot`. 2. **Image & Chart Agent** – Detects and interprets charts or images using OCR (`Tesseract`, `LayoutLMv3`). 3. **Data Interpretation Agent** – Uses LLM-based reasoning to structure extracted data into semantically meaningful form. 4. **Query Understanding Agent** – Interprets user questions and routes them to relevant sub-agents. 5. **Response Synthesis Agent** – Generates

coherent, context-rich answers combining multiple extracted data points. 6. **Storage Layer** – Vector database (e.g., Pinecone, FAISS) for semantic retrieval. 7. **Interface Layer** – A conversational chatbot or API endpoint for NLI queries.

6. Flow of Operations

1. User uploads PDFs into the system.
2. Ingestion Agent parses PDFs and stores vectorized representations.
3. Query Understanding Agent interprets user query.
4. Relevant data retrieved using semantic search.
5. Response Synthesis Agent compiles the answer.
6. Response displayed as text or visual summary (charts, tables, etc.).

7. Tools and Frameworks

- **Libraries:** LangChain, PyMuPDF, Camelot, Tesseract, Pandas, Matplotlib.
- **LLM APIs:** Gemini, OpenAI GPT, Claude.
- **Storage:** Pinecone, Chroma, FAISS.
- **Visualization:** Plotly, Matplotlib, Streamlit.
- **Deployment:** Google Cloud Run / Vertex AI for serving agents.

8. Evaluation Metrics

- **Accuracy** of extracted data.
- **Relevance** of responses to user queries.
- **Response latency** (in seconds).
- **Scalability** for multiple documents.
- **User satisfaction** in NLI responses.

9. Supportive Implementation Guide (with Hints)

Hint 1: PDF Ingestion Layer
- Use `PyMuPDF` for text extraction, `Camelot` for table parsing, and `pdf2image` for chart conversion.
- Store extracted segments in structured JSON for downstream processing.

Hint 2: Multi-Agent Coordination
- Implement each agent as a microservice communicating via APIs or message queues (e.g., Pub/Sub or RabbitMQ).
- Use a central **Controller Agent** to manage interactions between agents.

Hint 3: Embedding and Semantic Retrieval
- Convert parsed content into embeddings using models like `text-embedding-3-small` or Gemini Embeddings.
- Use a vector database to enable semantic matching between user queries and document content.

Hint 4: Query Understanding Agent
- Leverage LLM prompt engineering to classify query type (summary, trend analysis, lookup, etc.).
- Route accordingly to the interpretation or synthesis agent.

Hint 5: Response Generation and Validation
- The synthesis agent should construct responses combining text and visual summaries.
- Validate answers against numerical tables for accuracy.

Hint 6: User Interface and Visualization
- Build a simple Streamlit or Gradio app to allow file uploads and query input.
- Integrate Matplotlib or Plotly to visualize extracted data trends.

Hint 7: Scalability and Optimization
- Optimize parsing using concurrent workers or cloud-based batch processing.
- Cache vector embeddings to avoid reprocessing of the same PDFs.

Hint 8: Evaluation and Iteration
- Create test cases with known answers to benchmark response accuracy.
- Collect feedback from domain users and fine-tune the query-response logic.

---

10. Deliverables

- Functional multi-agent prototype for PDF ingestion and querying.
- Documentation on architecture and workflow.
- Demo app showcasing interactive data querying.
- Evaluation report with performance metrics.