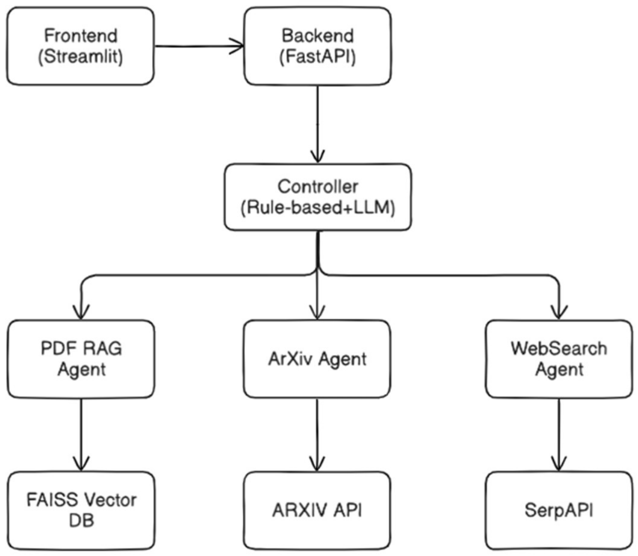
Project Report: Multi-Agent System

Prepared by: **Ayush Chaware**

1. Introduction
The Multi-Agent System (MAS) is an intelligent architecture that leverages specialized agents to handle diverse and complex user queries. Each agent operates independently with its own reasoning logic, and a central controller decides which agent(s) should handle each query. This design ensures scalability, modularity, and efficient use of computational resources.

Objective:
- Automate query handling across domains like documents, research, and general knowledge.
- Ensure that LLMs are used effectively by delegating specialized tasks.
- Provide real-time intelligent decision-making through rule-based and LLM-based routing.
- Deploy a production-ready multi-agent backend on Render using FastAPI.

2. System Architecture



The system consists of the following key components:

1. Frontend: A lightweight user interface that allows users to enter queries, upload PDFs, and view responses.
2. Controller (Backend - FastAPI): The brain of the system that receives the query and determines which agent(s) to invoke based on predefined rules and LLM-driven reasoning.
3. Agents: Independent service modules, each focusing on specific domains such as PDFs, research papers, or web knowledge.
4. LLM Integration: Large Language Models (e.g., via Groq API) used for intelligent decision-making and content synthesis.
5. -Integration Framework: LangChain (document processing, agent orchestration)
6. Database / File System: FAISS Temporary storage for user-uploaded PDFs and query logs.
7. Render Deployment Layer**:** Cloud infrastructure managing runtime, scaling, and HTTPS access.

Flow of Execution:
User → Frontend → Controller → Agent(s) → LLM → Aggregator → Response

This modular architecture ensures that new agents can be added easily without restructuring the core system.

Project Structure:

```
multi_agent_system/
├── backend/                # FastAPI Backend
│   ├── app/
│   │   ├── main.py          # FastAPI application entry
│   │   ├── api/             # API route handlers
│   │   │   ├── ask.py         # Main query processing
│   │   │   ├── upload.py      # PDF upload & status
│   │   │   └── logs.py        # System logs
│   │   ├── agents/          # Specialized AI agents
│   │   │   ├── controller.py  # Intelligent routing
│   │   │   ├── pdf_rag.py     # PDF document processing
│   │   │   ├── arxiv_agent.py  # Academic paper search
│   │   │   └── web_search.py   # Web search & synthesis
│   │   └── utils/           # Shared utilities
│   │     ├── logging_utils.py # Decision logging
│   │     ├── security.py    # Upload validation
│   │     └── backoff_utils.py # Retry mechanisms
│   ├── requirements.txt       # Python dependencies
│   ├── .env.example          # Environment template
│   └── data/               # Data storage
│     ├── uploads/          # Uploaded PDFs
│     └── vectorstore/       # ChromaDB storage
├── frontend/               # Streamlit Frontend
│   └── streamlit_app.py       # User interface
├── logs/                   # System logs
│   └── decision_logs.jsonl     # Agent routing decisions
├── sample_pdf/               # sample pdf to test RAG
│   └──.pdf file
├── .gitignore               # Git ignore patterns
├── REPORT.pdf                # Project report
├── render.yaml               # Render deployment config
└── README.md                  # This file
```

3. Logic of Agents

Each agent performs domain-specific tasks using a combination of deterministic rules and LLM reasoning.

3.1 PDF_RAG Agent

- Extracts text and metadata from uploaded PDF documents.
- Splits large documents into semantic chunks for token-efficient processing.
- Answers user questions based on document context using LLMs. Handles fallback for image-based PDFs.

3.2 ARXIV Academic Agent

- Designed to answer queries involving research papers or scholarly content.
- Searches academic databases or APIs for relevant information.
- Generates concise, citation-based summaries or explanations using LLM prompts.
- Ideal for research assistants, academic analysis, or literature surveys.

3.3 WEB_SEARCH Agent

- Handles general or open-domain queries.

- Fetches and summarizes relevant online information using APIs or web scraping.- Provides concise, factual responses filtered by LLM reasoning.

3.4 Aggregator Agent
- Merges responses from multiple agents when needed.
- Uses an LLM comparison prompt to select or merge the best answers.
- Example prompt: “Compare the following agent responses and produce the most accurate,concise, and well-structured final answer.”

4. Controller Logic (Rules + LLM Prompt)
The controller serves as the orchestrator. It first applies **rule-based logic** to determine which agent to invoke. If multiple agents are eligible, it uses an **LLM-based decision mechanism** to finalize routing.

Rule-Based Routing:

Condition	**Agent Invoked**
PDF uploaded	PDF Agent
Query contains 'research', 'paper', 'study'	ARXIV Academic Agent
General query “browse”, “latest”	WEB_SEARCH Agent

LLM Routing Prompt Example:
“You are an agent router. Choose ONE of: PDF_RAG, WEB_SEARCH, ARXIV.
User query: {text}
Respond with ONLY the agent name (one word):”

Prompt for Agents:
Each agent uses tailored prompts to ensure domain relevance, accuracy, and structured output.

For instance,
the **PDF_RAG** Agent may receive prompt:
“You are a PDF analysis assistant. Use the following extracted text and the user query to provide a well-structured answer with cited sections.”

WEB_SEARCH Agent prompt:
“ You are a helpful AI assistant providing comprehensive, accurate answers based on current web search results.
User Query: {query}
Search Results from Google:
{formatted_results}
Instructions:
1. Provide a comprehensive, well-structured answer to the user's query
2. Synthesize information from multiple sources
3. Include specific details, dates, facts, and figures when available
4. Organize information with clear sections using markdown headers (##, ###)
Provide your detailed answer:”

ARXIV Agent Prompt:
“You are an AI research assistant analyzing recent academic papers from ArXiv.
User Query: {query}
ArXiv Papers Found ({len(papers)} papers):
{papers_text}
Please provide a comprehensive, well-structured analysis with the following sections:
Overview
Provide a 2-3 sentence summary of the current research landscape in this area based on these papers.

Key Papers & Contributions
For each significant paper (top 3-5),
Research Trends & Themes
Identify common patterns, methodologies, or emerging directions across these papers.
Notable Researchers
List prominent authors who appear across multiple papers or are from well-known institutions. Format Guidelines:
- Use markdown headers (##, ###)
- Use **bold** for paper titles and key terms
- Use bullet points for lists
- Include ArXiv IDs in format: [2510.05102]
- Keep the analysis comprehensive but concise
Provide your detailed analysis:”

5. Design Trade-offs
Every design decision comes with trade-offs between performance, scalability, and maintainability.

- Latency vs. Accuracy: Running multiple agents increases accuracy but introduces higher response time.
- Cost vs. Coverage: More LLM and API calls mean higher operational cost.
- Rule-Based Simplicity vs. LLM Intelligence: Static rules are predictable, but LLM routing offers adaptability.
- Scalability vs. Maintainability: Adding agents is easy, but managing inter-agent communication can become complex.
- Prompt Brittleness**:** The system’s reliability depends heavily on prompt design and version control.

6. Deployment Notes
The project is deployed on **Render** as a FastAPI web service.

Steps:
1. Containerized the FastAPI backend using Docker.
2. Linked the GitHub repository to Render for CI/CD integration.
3. Configured environment variables for API keys and secrets.
4. Set the web service to auto-deploy on main branch updates.
5. Utilized Render’s autoscaling and HTTPS configuration.

Challenges & Solutions:
- Cold Start Delay: Solved by keeping the service active via periodic health pings.
- Timeouts: Increased timeout limit for long-running PDF processing.
- Memory Constraints: Implemented chunking for large PDF documents.
- Logging**:** Added detailed logs for agent decisions and LLM responses for better debugging.

Outcome:
A stable, cloud-deployed, production-grade FastAPI service that can intelligently route and answer queries in real-time.

7. Conclusion
The Multi-Agent System successfully integrates rule-based decision-making, LLM routing, and specialized agents into one cohesive framework. It is modular, extensible, and cloud-deployable, making it a scalable foundation for future AI-driven applications.

Future Enhancements:.
- Integrate caching and memory for context persistence.
- Add more domain-specific agents (finance, healthcare, etc.).

- Implement a frontend dashboard for monitoring agent performance.

This project demonstrates the power of combining rule-based AI orchestration with modern LLM reasoning to build practical, intelligent systems.