# Multi-Agent Research Assistant — Project Report

## 1. Introduction

This project implements a modular Multi-Agent System designed to automate research assistance tasks. It integrates multiple AI agents such as PDF RAG Agent, Web Search Agent, ArXiv Agent, and a central Controller Agent. The system leverages LLMs (via Groq API) for reasoning and summarization. The user can upload PDFs, ask research-related queries, and retrieve summarized answers from documents, the web, or recent ArXiv papers.

## 2. System Architecture

The architecture follows a hybrid design combining rule-based routing and LLM-driven reasoning. Each agent handles a specific function, coordinated by the Controller Agent.

Figure 1: High-level Architecture of the Multi-Agent System. (Diagram omitted in text version)

## 3. Agent Interfaces

|  |  |  |
| --- | --- | --- |
| Agent | Description | Key Functions |
| Controller Agent | Central decision-maker that routes queries based on rules or LLM reasoning. | Routes queries → selects agents → merges results → logs traces. |
| PDF RAG Agent | Handles uploaded research papers or domain-specific PDFs using FAISS embeddings. | Text extraction, chunking, vector search, and summarization. |
| Web Search Agent | Retrieves live data using SerpAPI or Gemini-based web access. | Fetches recent information, summarizes top web results. |
| ArXiv Agent | Queries ArXiv for recent or relevant scientific publications. | Fetches titles, authors, and abstracts of recent papers. |

## 4. Controller Logic

The Controller Agent decides which agents to activate based on the query context. The routing follows a hybrid approach — predefined rules handle simple cases, while LLMs assist in ambiguous queries.

Rule-based Routing Examples:

• If query contains 'summarize' + uploaded PDF → use PDF RAG Agent.

• If query contains 'recent papers' or 'arxiv' → use ArXiv Agent.

• If query mentions 'latest' or 'news' → use Web Search Agent.

• Otherwise, default to combined ArXiv + Web Search results.

## 5. Trade-offs & Challenges

• LLM Dependency: While Groq and Gemini APIs provide strong reasoning, they introduce rate limits and dependency on API availability.  
• Latency vs Accuracy: Combining multiple agents increases inference time but improves contextual relevance.  
• Deployment Complexity: Maintaining communication between FastAPI and Streamlit on cloud environments like Render or HF Spaces requires careful port management.  
• Scalability: FAISS-based RAG indexes are lightweight but not suitable for extremely large document sets.

## 6. Deployment Notes

The project is deployed on Render as a web service running both the FastAPI backend and Streamlit frontend. Render automatically detects the application and runs it based on the provided start command in the Render dashboard.

Deployment Steps Summary:

Push the complete project to GitHub with all dependencies listed in requirements.txt.

Log in to Render and create a new Web Service.

Connect your GitHub repository and choose the branch (usually main).

In the Root Directory, specify the folder containing your backend code (e.g., / or /backend depending on structure).

In the Start Command, enter:

uvicorn backend.main:app --host 0.0.0.0 --port 8000

Add all required environment variables (like GROQ\_API\_KEY, SERPAPI\_KEY, etc.) under Render → Environment → Environment Variables.

Deploy the app and wait for it to build successfully.

Copy the Render deployment URL and update your frontend (app.py) BACKEND\_URL to this Render URL.

Redeploy if necessary to ensure both backend and frontend communicate properly.

This setup allows users to access the full multi-agent system online, including PDF upload, query handling, and logs view, through the deployed Render URL.

## 7. NebulaByte PDF Dataset

Five domain-specific PDFs were curated from NebulaByte Technologies’ internal dialogues, technical notes, and reports. These were converted into research-style documents and stored in `sample\_pdfs/` for demonstration of retrieval-augmented generation (RAG). Each PDF simulates realistic research contexts for energy optimization, AI ethics, and automation systems.

## 8. Conclusion

This multi-agent research assistant demonstrates how modular AI systems can automate literature review, knowledge retrieval, and summarization. The combination of rule-based routing and LLM-powered summarization ensures reliability, adaptability, and transparency in research automation workflows.