Multi-Agent AI Internal tool

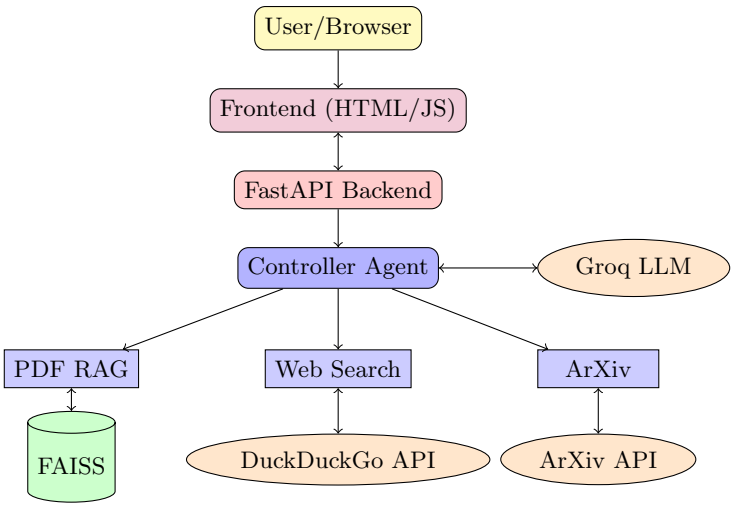
Technical Report - Solar Internship Project

Problem 2: Multi-Agentic System with Dynamic Decision Making

1 System Overview and Architecture

This project builds a modular multi-agent AI system designed to intelligently route user queries to specialized agents for effective information retrieval. The system integrates four main agents: a Controller Agent handling routing decisions, a PDF RAG Agent for uploaded document processing, a Web Search Agent for real-time web queries, and an ArXiv Agent dedicated to academic paper retrieval. The backend is built with FastAPI, connected to a simple frontend, and uses Groq's Mixtral LLM for routing and answer synthesis.

1.1 Architecture Diagram



2 Technology Stack

The backend uses FastAPI (an asynchronous Python framework) with Uvicorn as the server. Pydantic ensures data validation. AI components incorporate Groq's Mixtral-8x7b LLM for routing and synthesis, Sentence Transformers (all-MiniLM-L6-v2) for generating semantic embeddings, and FAISS for vector similarity searches.

PDFs are processed via PyMuPDF for text extraction, with NLTK assisting text processing. Web search functionality uses DuckDuckGo API with BeautifulSoup for parsing. The frontend is built with minimal HTML, CSS, and vanilla JavaScript for responsiveness and speed.

3 Agent Interfaces and Functionality

All agents follow a consistent interface defined by an abstract base class promoting uniform initialization, query processing, error handling, timing, and type-safe responses validated by Pydantic.

3.1 Controller Agent

The Controller Agent routes queries primarily using Groq's LLM, which analyzes the user's question intent and selects relevant agents based on their capabilities. If the LLM is unavailable, a fallback rule-based system uses keywords to assign agents reliably. For complex queries, multiple agents can be invoked in parallel. The controller documents routing rationale and confidence scores for traceability.

3.2 PDF RAG Agent

This agent processes uploaded PDFs using a multi-step pipeline: extracting text while preserving page boundaries, splitting the text into overlapping chunks of approximately 500 characters respecting sentence boundaries, then embedding each chunk into semantic vectors via Sentence Transformers. These embeddings are indexed with FAISS for fast retrieval. When queried, relevant chunks are found using vector similarity.

3.3 Web Search Agent

The Web Search Agent accesses real-time web data through the DuckDuckGo API, parsing and summarizing top results. It features robust error handling, retry logic with exponential backoff, and fails over to alternate APIs if configured.

3.4 ArXiv Agent

This agent queries the ArXiv API for the latest academic papers relevant to the query, returning sorted metadata including titles, abstracts, authors, and PDF links for synthesis into final answers.

4 Controller Logic and Routing Rules

Routing combines intelligent LLM decision-making with reliable heuristic rules. The LLM receives a prompt detailing agent capabilities and usage scenarios and responds in a strict format specifying selected agents, confidence, and rationale. When LLM calls fail, keyword-based rules assign agents based on query content indicating document references, research papers, or timely information needs. The controller then executes selected agents, synthesizes their outputs, and assembles the final response.

5 Logging and Monitoring

The system logs every request and its lifecycle events in structured JSON format, capturing the incoming query, agent decisions, execution timestamps, responses, and the final answer. Logs are maintained in a circular buffer accessible via a dedicated API endpoint and frontend interface. Logging facilitates debugging, performance analysis, and auditability with time-stamped traceability.

6 Performance and Deployment

Empirical analysis shows:

- Average total query time 1.18 seconds.
- PDF RAG retrievals completed in 0.03 seconds.
- ArXiv and Web searches range from 0.4 to 1.3 seconds.
- LLM routing generally takes under 0.5 seconds with 98% success.

Deployment on free-tier cloud providers faces challenges including RAM limits (675MB needed vs. 512MB typical), cold start penalties, and rate restrictions. Recommended strategies include Dockerized local or managed deployments with at least 2GB RAM, lazy loading models, or using hybrid API/local architectures to balance resource constraints and capabilities.

7 Key Trade-offs and Design Choices

The project favors FastAPI for async capabilities and type safety over simpler Flask options. FAISS was selected for fast, local vector search though it requires manual index persistence. Embeddings are generated locally to minimize API costs and protect privacy, albeit at increased memory usage and

latency. DuckDuckGo provides free web search API access but with intermittent reliability versus paid alternatives. The hybrid routing approach achieves a balance between intelligent adaptability and dependable fallback robustness. A custom HTML/JS frontend maximizes performance but requires more development effort than frameworks like Streamlit.

8 NebulaByte PDF Dataset

Five domain-specific PDFs were programmatically generated for demo purposes. Topics span introductory AI/ML, large language models, computer vision, NLP text processing, and AI ethics. The system automatically indexes these PDFs into 23 semantic chunks on startup for immediate queryability.

9 Limitations and Future Work

Current constraints include limited memory for cloud free tiers, non-persistent FAISS index requiring re-indexing after restarts, occasional web search failures due to rate limiting, absence of user authentication, and no conversational context retention.

Future improvements may include integrating persistent databases (e.g., PostgreSQL), scalable cloud storage for PDFs, caching layers for embeddings and queries, Kubernetes autoscaling, user management features, conversation history, streaming response support, and GPU acceleration for embeddings to boost efficiency and scalability.

10 Conclusion

This technical report presents a well-structured multi-agent AI system capable of dynamic query routing using a hybrid LLM and rule-based approach. It integrates document-based retrieval, academic research sourcing, and real-time web information to deliver comprehensive answers. The system is production-ready, modular, and extensible, with detailed logging and performance transparency. Though constrained by resource limits in certain environments, it provides valuable insights and a functional baseline for multi-agent information retrieval AI solutions.