Report: My Multi-Agent Intelligence Bot

Somesh Jaiswal
CO23BTECH11022

This final report provides a strategic and technical evaluation of my platform. It integrates recent insights on transparency, privacy, and competitive positioning to define why this product stands out in a crowded AI market.

1 Architecture and Technology

My platform is built on a Stateful Multi-Agent System (MAS) paradigm. Unlike simple "chat wrappers," this is a deep-tech orchestration layer designed for high-autonomy tasks.

Orchestration Engine (LangGraph): The "nervous system." It manages state transitions, ensuring that my preferences or a PDF's summary persists across different nodes without "forgetting."

The Data Plane: Vector Store: FAISS with HuggingFaceEmbeddings provides a local semantic search engine.

Persistence: SQLite enables long-term memory, allowing the system to resume complex research tasks across multiple sessions.

The Compute Layer: A hybrid approach using high-throughput models (Llama-3 via Groq) for reasoning and specialized vision models (Gemini) for technical diagram generation.

Streamlit for Frontend Part: I chose Streamlit to serve as the interactive command center of my platform. It allows for a reactive, Python-native interface that handles complex states like file uploads (for RAG) and real-time streaming of agent "thoughts." By using Streamlit's session state, I synchronize the UI with the underlying LangGraph threads, providing a seamless transition between the three specialized agents.

2 Critical Analysis

Modularity over Monoliths: By separating the "Research," "Memory," and "RAG" agents, I avoid Instruction Drift. Each agent has a specialized system prompt, ensuring that the RAG bot doesn't start hallucinating web data when it should be reading my private PDF.

Making it Microservice: I designed the agent architecture to mimic a microservices ecosystem. Each agent (Memory, RAG, Blog) operates as an independent logical unit within the LangGraph. This modularity means I can update the RAG tool's embedding model without breaking the Blog Agent's research logic. It ensures that the system is resilient and that each "service" can be optimized for its specific domain—whether that is fast conversational retrieval or deep-dive research.

Cognitive Load Reduction: The Orchestrator-Worker pattern in the Blog Agent mirrors human project management. It breaks a complex 2,000-word blog into 5 manageable "tasks," writes them in parallel, and merges them—significantly reducing the time a user spends on manual synthesis.

3 Originality and Advantages

Fixing Today's LLM Issues: The primary problems with current LLMs are hallucinations and data privacy. Traditional models often "fill in the gaps" with plausible but false information when they lack context. I was motivated to fix this by building a product that prioritizes grounding. By using RAG, my platform forces the LLM to look at my data first, treating it as the "Source of Truth." This shifts the AI from a creative writer to a factual synthesizer, ensuring that the output is reliable and the data remains within my controlled environment.

Transparency as a Feature: While others offer "Black Box" AI, my tool exposes the Step-by-Step Workflow. Users see exactly what the agent is searching for, which tools it is calling, and how it is reasoning. This builds high trust.

Local Sovereignty: Running logic locally (or via private API keys) ensures that sensitive corporate or personal data is not being used to train a global model.

Accuracy: I enforce a strict Grounding Policy. If a fact isn't in the provided evidence (PDF or Search), the agent is instructed to state "Not found," rather than making it up.

4 Comparison with Industry Leaders

Feature	GPT / Gemini	Perplexity	My Multi-Agent Platform
Long-term History	Session-based / Basic	Thread-based	Persistent Fact Extraction
Research Tools	Integrated (GPT Search)	Citation-focused Search	Autonomous Pipeline (Search → Plan → Write)
Privacy Model	Cloud-centralized	Cloud-centralized	Local-First / Private Knowledge Base
Knowledge Base	Fixed / Web	Web-only	Private PDF RAG + Real-time Web Hybrid
Workflow Transparency	Minimal (hidden logic)	Medium (citations)	Full Traceability (Step-by-step monitoring)
Custom Knowledge	Custom GPTs (Cloud)	Collections (Cloud)	Private Local Document Ingestion

The Key Differentiator: While GPT and Perplexity are "Information Retrievers," my product is a "Knowledge Architect." It allows a user to "own" their intelligence by grounding the AI in their own files while keeping the entire process transparent and private.

5 Impact It Creates

Professional Work: Reduces "research-to-draft" time from hours to minutes.

User Empowerment: The "Step-by-Step" view serves as a Learning Tool. Users don't just get an answer; they see the process of how to research a topic, teaching them better information literacy.

Domain Specialization: A major problem with general LLMs is that they are "jacks of all trades but masters of none." By using RAG, I make searching not restricted to a general task. I can specialize my model in any domain—be it bioinformatics, legal research, or thermodynamics—simply by feeding it the relevant documents. This transforms the general LLM into a Domain Expert that understands specific terminologies and niche data that

general models might ignore or get wrong.

Regulatory Compliance: Meets the needs of legal and financial sectors where data must stay within a private or local environment.

6 Potential Business Model & Scaling

Currently, my product is a basic, high-utility prototype. It is not yet fully scaled or deployed due to resource constraints. However, if scaled properly with a scalable backend and dedicated API endpoints, it could evolve into a robust production-grade tool:

Scalable Infrastructure: Moving from a local script to a FastAPI-powered backend would allow my agents to handle thousands of concurrent requests. By exposing my LangGraph workflows as APIs, I can allow other platforms to "rent" my research or RAG capabilities.

SaaS Tiers: Individual: Personal research and local PDF management. Pro: Advanced research runs and image generation. Enterprise Scaling: Deploying "Private Nodes" for companies. Instead of a single user, a whole team can query a shared "Internal Knowledge Base" (RAG) that acts as a Corporate Librarian.

7 Relevance in Different Areas

RAG (Retrieval-Augmented Generation): Essential for Legal (case files), Healthcare (literature), and Academic Research (textbooks).

Context Variability: The beauty of my platform is its adaptability. Chatbots are used everywhere, but most are "static." My platform's ability to handle context variability means it can be adopted anywhere—from an onboarding assistant in a tech firm to a literature synthesizer in a lab. Because the "Memory" and "RAG" layers update dynamically, the platform effectively "re-skins" its intelligence based on the user's current project or industry.

8 Disadvantages & Limitations

While my platform offers significant advantages, there are current trade-offs:

Limited scalability: Due to resource constraints, the system is not well-scaled. As a result, it experiences higher latency and lower throughput compared to a production-grade

deployment.

API usage limitations: The project relies on free/open-source model APIs for embeddings and responses. These services impose daily usage limits, rate limits, and context/token restrictions, which can sometimes lead to errors. While paid API plans could significantly improve reliability and performance, the goal was to keep the system accessible to general users without additional costs.

No full cloud deployment: The application is not deployed on cloud platforms such as AWS or GCP due to cost considerations.Therefore, users cannot access it via a public IP or domain. Instead, it is currently designed to run locally using Streamlit.

Limited feature set: At present, the system includes three main components — a chatbot, a RAG tool, and a research assistant. Future plans include expanding functionality, such as adding image generation capabilities and further performance optimizations.

Accuracy Variance: While RAG reduces hallucination, the output quality is still dependent on the quality of the ingested PDF and the chunking strategy. Similarly for research agent, accuracy depends on the quality of system prompts, which I used are not much good, because more better prompt requires more resources.

9 Final Verdict

My platform transitions AI from a "Chatbot" to a "Digital Colleague." By focusing on Privacy, Transparency, and Multi-Agent Orchestration, I have created a tool that respects the user's data while maximizing their cognitive output.

I am thankful to professor **G - Narahari Sastry** for this opportunity to work on an application which has real world application and helps me to learn many things.

THE END

10 How to Run the Application

Follow the steps below to set up and run the application locally.

10.1 Step 1: Clone the Repository

Create a new folder and clone the GitHub repository:

```
git clone https://github.com/SomeshIITH/Multi-Agent-Chatbot
```

cd Multi-Agent-Chatbot

10.2 Step 2: Install Dependencies

Install the required Python packages:

```
pip install -r requirements.txt
```

10.3 Step 3: Configure Environment Variables

Create a file named `.env` in the project root directory and add the following placeholders:

```
GROQ_API_KEY = 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
TAVILY_API_KEY="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"
```

10.4 Step 4: Run the Application

Launch the Streamlit frontend:

```
streamlit run mainfrontend.py   OR    python -m streamlit run mainfrontend.py
```

The application will open in your default browser.

10.5 All 3 Services can work independently , to run them

For Simple Chatbot (Use python -m before streamlit if it not works)

```
streamlit run simpleChatbotfrontend.py
```

For RAG Agent

```
streamlit run ragChatbotfrontend.py
```

For Research Blog Agent

```
streamlit run blogChatbotfrontend.py
```