**Project Development Report**

**LinkedIn Career Coach AI – Approach, Challenges, and Solutions**

**Overview**

This document outlines the step-by-step approach taken to develop a multi-agent, LLM-powered system for LinkedIn profile optimization, job fit analysis, and personalized career guidance. It highlights the technical decisions made throughout the project, the core implementation strategy, and the challenges encountered during development, along with how they were addressed.

**1. Approach and Design Strategy**

**a. Initial Exploration and Model Setup**

The development initially began using OpenAI’s models with LangChain in a local VS Code environment. However, multiple dependency conflicts arose during integration, particularly around LangChain’s evolving architecture. As a result, the development was moved to Google Colab, which offered better compatibility. However, Colab’s default OpenAI token management caused repeated quota exhaustion during testing.

After exploring Gemini’s Pro and other Gemini variants, the final model selection settled on **Gemini 1.5 Flash**. This offered a stable balance between capability and token limits, making it suitable for rapid prototyping and multi-agent inference workflows.

**b. Data Source Strategy and LinkedIn Profile Ingestion**

The system initially aimed to scrape live LinkedIn profiles using Apify’s API and other custom scrapers (e.g., Selenium, BeautifulSoup-based scripts). However, due to LinkedIn’s aggressive anti-bot measures and API rate limits, most public profiles—including the developer’s own—could not be scraped successfully. Even approaches involving authentication cookies and session injection failed due to heightened security on LinkedIn's side.

To overcome this, a practical pivot was made to allow **direct upload of LinkedIn profile PDFs**, which all users can download from their accounts. This provided complete, structured data in an accessible format. PyPDF2 was used for parsing the PDF and extracting key profile sections (About, Experience, Skills, Education, etc.), forming the base context for downstream agents.

**2. Functional Breakdown and Multi-Agent Architecture**

The architecture was designed to modularly separate responsibilities among individual intelligent agents, eventually composing them into a pipeline using LangGraph. Each agent was given access to specific context relevant to its task to maximize efficiency and relevance.

**a. Profile Analysis Agent**

* **Input**: Extracted LinkedIn profile data
* **Processing**: Prompt-based section-wise evaluation using Gemini
* **Output**: Profile gaps, quality assessments, and suggestions
* This agent served as the foundation for all downstream enhancement and guidance tasks.

**b. Job Fit Analysis Agent**

* **Input**: Profile data, job title (user-specified), real-time job descriptions
* **Processing**: Live scraping of remoteok.com to retrieve top 8–10 job listings for the given role
* **Logic**: Matching full and partial job title keywords; only full matches used for analysis
* **Output**: Match score, 3 strengths, and 3 weaknesses; partial matches shown as alternatives
* Several other job boards like Indeed and Glassdoor were explored but ultimately abandoned due to security walls, with RemoteOK providing the best scraping accessibility.

**c. Content Enhancement Agent**

* **Input**: Profile data, profile analysis output, top 3 scraped JDs
* **Processing**: Content rewriting aligned to job-market expectations
* **Output**: Rewritten sections customized to current job requirements
* This was implemented as a lightweight RAG-style prompt construction, where all three contexts were provided to Gemini for content enhancement.

**d. Career Coach Agent (Router Agent)**

* **Input**: All above agent outputs + user query
* **Logic**: Determines what context is required based on query type (e.g., email writing, career switch, improvement plan)
* **Functionality**:
  + Crafts emails to recruiters with profile highlights
  + Suggests career paths for a given title
  + Identifies transferable skills for role switching
  + Recommends specific upskilling resources
* This agent simulates a decision-making mentor and conditionally invokes contextual knowledge from other agents.

**3. LangGraph Implementation**

To coordinate agent execution and ensure a consistent internal state across all steps, the system uses LangGraph:

* A ProfileState was defined to hold profile data, analysis, job title, job matches, enhancement results, and user queries.
* Each agent was wrapped as a RunnableLambda with defined edges for linear and conditional execution.
* MemorySaver was used to ensure lightweight memory persistence for context tracking.
* The LangGraph was compiled into a directed flow graph, with entry and exit points aligned to agent invocation order.

**4. Frontend Interface and UI Development**

**a. Initial Attempts via Colab + Streamlit**

Streamlit integration was attempted directly within Colab using tunneling solutions like pyngrok. However, output routing failures and file system inconsistencies delayed proper rendering.

**b. Final Integration in VS Code**

The backend was migrated back to VS Code. Issues related to google.colab imports were resolved by rewriting the upload logic using Streamlit's built-in file uploader. Two separate Python files were maintained:

* agents.py: LangGraph logic and agent definitions
* app.py: Streamlit UI logic

Each core agent was mapped to a separate tab in the Streamlit app:

* Tab 1: Profile Analysis
* Tab 2: Job Fit Analysis
* Tab 3: Content Enhancement
* Tab 4: Career Guidance

Inputs (job title, user question, PDF upload) were modularly exposed, and outputs were rendered only when respective buttons were clicked.

**5. Challenges and Solutions**

| **Challenge** | **Solution** |
| --- | --- |
| LinkedIn scraping limitations | Switched to PDF-based input |
| Model quota limits (OpenAI, Gemini Pro) | Settled on Gemini 1.5 Flash |
| Colab + Streamlit tunneling failure | Shifted to VS Code and decoupled backend/frontend |
| Agent context overload in career guide | Used conditional routing logic based on question type |
| Job scraping restrictions | Chose RemoteOK for scraping ease and relevance |

**6. Technical Highlights**

* Agent-based design ensures modularity, scalability, and maintainability
* Prompt engineering optimized for section-specific and task-specific outputs
* LangGraph used to simulate decision-making with persistent state
* End-to-end LLM pipeline with real-time scraping and structured enhancement
* All user-facing interactions built with Streamlit for easy accessibility

**Conclusion**

This project successfully integrates modern LLM infrastructure with practical data ingestion and interaction methods. All project goals were met, including multi-agent orchestration, job-contextual content enhancement, and a chat-based career guide system. The resulting application demonstrates strong use of Gemini models, real-time job scraping, and streamlined UX — packaged as a functional career assistant platform.