## 1. Introduction

This project implements an automated Know Your Customer (KYC) resume verification system using a Large Language Model (LLM) agent integrated with external social network tools. The purpose of the system is to validate candidate CV information by comparing it against real social media profiles from LinkedIn and Facebook, and then generate a credibility score reflecting the authenticity of the resume.

This system automates the entire process using an intelligent agent capable of extracting identity information, searching external databases, reasoning about discrepancies, and producing structured verification results. The system leverages the LangGraph ReAct agent architecture, Google Gemini LLM, and MCP-based external tools.

## 2. System Architecture Overview

The system consists of three main components: the Language Model, the Tool Integration Layer, and the Verification Execution Loop.

First, the Language Model acts as the reasoning engine. It is responsible for extracting key information from resumes, invoking external tools, comparing retrieved profiles, and assigning verification scores. The Gemini model is configured with temperature set to 0 to ensure consistent and deterministic outputs, which is important for scoring reliability.

Second, the Tool Integration Layer connects the system to external data sources. MCP tools provide access to LinkedIn and Facebook profile search and retrieval functions. These tools allow the agent to retrieve real-world identity information, which forms the basis for verification.

Third, the Verification Execution Loop processes resumes one by one. Each resume is sent to the agent, analyzed, and assigned a score based on the consistency between the resume and social media profiles.

## 3. Agent Creation and Reasoning Mechanism

The agent is created using the ReAct (Reasoning and Acting) framework, which allows the model to reason and use tools iteratively.

Important code example 1 (Agent creation):

kyc_agent = create_react_agent(llm, tools, prompt=system_prompt)

This line creates an intelligent verification agent by combining three essential elements: the language model (llm), the external verification tools (tools), and the system

instructions (system_prompt).

The system prompt defines the verification procedure, including extracting resume information, searching LinkedIn and Facebook, retrieving profiles, comparing identity information, and assigning a credibility score. The agent follows this structured reasoning process autonomously.

The ReAct framework enables the agent to alternate between reasoning and tool usage. For example, the agent first extracts a candidate's name, then searches LinkedIn, retrieves profiles, evaluates matches, and finally produces a score. This reasoning-action cycle improves accuracy and makes the system capable of handling complex verification tasks.

**4. Resume Processing and Agent Execution (Core Z Section)**

This section represents the most critical part of the entire system. It is responsible for sending resumes to the agent, receiving the agent's output, extracting structured results, and converting them into usable verification scores.

Important code example 2 (Agent execution):

```python
agent_response = await kyc_agent.ainvoke({
    "messages": [HumanMessage(content=user_instruction)]
})
raw_agent_output = agent_response["messages"][-1].content
```

This code asynchronously sends the resume content to the agent and retrieves the agent's response. The ainvoke() function executes the agent and returns a sequence of messages generated during reasoning. These messages may include intermediate reasoning steps, tool calls, and the final verification result.

The system extracts the final message using [-1], which contains the agent's final answer. This final answer is expected to include a JSON object containing the verification score and reasoning explanation.

However, the output format from large language models is not always consistent. The response may contain additional text, formatting, or markdown symbols. Therefore, additional processing is required to safely extract the structured JSON result.

**5. JSON Extraction and Score Parsing (Most Critical Z Logic)**

The most important function of the system is extracting the verification score from the agent's response. Since the agent is instructed to output the result in JSON format, the system uses regular expressions and JSON parsing to extract the score reliably.

Important code example 3 (JSON extraction and parsing):

```python
json_pattern_match = re.search(r'```json\n(.*?)\n```', final_agent_output, re.DO
if json_pattern_match:
    verification_result = json.loads(json_pattern_match.group(1))
    score = float(verification_result.get("score", 0.5))
```

This code searches for the JSON block inside the agent's response. The regular expression identifies content enclosed within json and markers. The extracted JSON string is then converted into a Python dictionary using json.loads().

The score field is extracted and converted into a floating-point number. If the score field is missing, a default score of 0.5 is assigned. This ensures that the system remains stable even if the agent produces incomplete output.

This step is essential because it transforms unstructured language model output into structured numerical data that can be used for evaluation and reporting.

### 6. Robustness and Error Handling

One of the key strengths of this implementation is its robustness. Since language model outputs may vary in format, the system includes multiple safeguards.

First, it handles different output formats by converting lists, dictionaries, or strings into a unified text format before parsing.

Second, it uses fallback parsing methods in case the JSON block cannot be found using regular expressions.

Third, exception handling ensures that system failures do not interrupt the entire verification process. If an error occurs, a default score of 0.5 is assigned, and the system continues processing the remaining resumes.

This fault-tolerant design makes the system suitable for real-world deployment scenarios where reliability is critical.

### 7. Verification Scoring Logic

The verification score reflects how closely the resume matches real social media profiles. The score ranges from 0.0 to 1.0, with higher scores indicating greater authenticity.

A score close to 1.0 indicates that the resume information matches the candidate's LinkedIn or Facebook profile accurately. Minor inconsistencies, such as outdated job titles, may result in slightly lower scores between 0.5 and 0.9.

Scores below 0.5 indicate significant discrepancies, such as incorrect employment history or education credentials. Extremely low scores suggest that the identity may be fabricated or unverifiable.

This scoring system allows automated filtering of suspicious resumes and improves recruitment security.