FTEC5660 Agentic AI for Business and FinTech

Homework 02 (Part 1) — Report

---

1. System Architecture and Design Decisions

My system is basically a pipeline with three stages: PDF parsing, social media data retrieval, and LLM-based comparison. I didn't go with a complicated multi-agent architecture because honestly the task is pretty straightforward — we just need to pull info from CVs, look up the same person on LinkedIn/Facebook, and check if things match up.

For PDF parsing, I used the MarkItDown library, which converts PDF files into markdown-formatted text. The output is a bit messy (lots of table formatting with | separators), but it keeps the content readable enough for the LLM to understand.

For the LLM, I went with DeepSeek (deepseek-chat) through the OpenAI-compatible API. I set temperature to 0 so the output is deterministic and consistent across runs. I considered using Gemini but ended up sticking with DeepSeek since it worked fine in my tests and the assignment says it doesn't affect scoring.

The MCP connection is set up using langchain_mcp_adapters, pointing to the course MCP server at ftec5660.ngrok.app. The server provides 6 tools total — 3 for Facebook (search users, get profile, get mutual friends) and 3 for LinkedIn (search people, get profile, get interactions). In my current implementation, I mainly rely on 4 of them: the search and profile retrieval tools for both platforms. I didn't end up using get_facebook_mutual_friends or get_linkedin_interactions because the core verification can be done just by comparing profile data directly.

One design choice I want to mention is the scoring scheme. Since the evaluation uses 0.5 as a hard threshold, I made the LLM output either 0.85 (for valid CVs) or 0.15 (for problematic ones), leaving a big margin on both sides. I didn't want a situation where the score lands at like 0.52 or 0.48 and gets misclassified due to small variations in the LLM response.

2. Agent Workflow and Tool Usage Strategy

The verification process for each CV works as follows:

Step 1 — Name Extraction. I parse the first line of the converted markdown text to get the candidate's name. The logic splits the line by |, strips whitespace, and joins the first one or two non-empty tokens depending on whether the first token already contains a space. For example, | Wei Zhang | ... | gives "Wei Zhang" directly, while | Rahul | Sharma | ... | gets joined into "Rahul Sharma". This is kind of hacky but it worked for all 5 sample CVs so I kept it.

Step 2 — LinkedIn Lookup. Using the extracted name, I call search_linkedin_people with a limit of 20 results. Then I take the top 5 search results and call get_linkedin_profile on each of them to get their full profile data (work history, education, skills, etc). I fetch 5 instead of just 1 because sometimes the top result isn't the right person, especially for common names like "Rahul Sharma".

Step 3 — Facebook Lookup. Same idea — search_facebook_users first, then get_facebook_profile on the top 5 results. Facebook data gives additional info like hometown, current job/company, bio, and posts, which can help cross-check things.

Step 4 — LLM Comparison. I feed the original CV text plus all the retrieved LinkedIn and Facebook profiles into the LLM in a single prompt. The prompt tells the LLM to act as a "CV fraud detector" and specifically asks it to:

- Find which LinkedIn profile is the best match for the candidate
- Check if the company names on the CV match the LinkedIn work history
- Check if the school name matches
- Check if the degree field matches

If all three match, it outputs 0.85. If any of them differ, it outputs 0.15.

I focused on these three checks because they're the most factual and verifiable items on a CV. Things like job titles might have slight wording differences that don't actually indicate fraud (e.g., "Engineer" vs "Software Engineer"), but company names, school names, and degree fields should be pretty exact. If someone claims they worked at Microsoft but their LinkedIn says Goldman Sachs, that's a clear red flag.

The tool usage is sequential rather than agentic in the ReAct sense — I didn't implement a loop where the LLM decides which tool to call next. I found that a fixed pipeline was more reliable for this task because the steps are always the same: search → get profiles → compare. Letting the LLM decide the tool order sometimes led to it skipping steps or calling irrelevant tools in my early tests.

3. Sample Verification Results

I ran the system on the 5 provided sample CVs. Here are the results:

CV	Candidate	Score	Prediction	Ground Truth	Correct?
CV_1	John Smith	0.85	Valid (1)	1	Yes
CV_2	Minh Pham	0.85	Valid (1)	1	Yes
CV_3	Wei Zhang	0.85	Valid (1)	1	Yes
CV_4	Rahul Sharma	0.15	Problematic (0)	0	Yes
CV_5	Rahul Sharma	0.15	Problematic (0)	0	Yes

Final accuracy: 5/5 = 100%

For CV_1 to CV_3, the LLM confirmed that the key information (employer names, university names, degree fields) on the CVs were consistent with what was found on their LinkedIn profiles, so they were scored as valid.

For CV_4, there are several suspicious points. The CV lists a "Senior Engineer" role at Microsoft with dates "2021–2027", but 2027 hasn't happened yet, which is obviously wrong. The candidate claims a PhD in Legal Studies from Tsinghua University, but the CV lists skills like "Web3, Machine Learning, Quantum Computing" — these don't really align with a legal background. When compared against LinkedIn data, these inconsistencies were caught.

For CV_5, the candidate (also named Rahul Sharma) claims to be an "AI Professional" with roles at EY, StartupXYZ, DataForge, and UrbanFlow. The "Senior Analyst" role at DataForge is listed as "2016 – Present", which overlaps with other positions in a way that seems off. The PhD from University of Tokyo is listed with just "2012" which is vague. The LLM flagged discrepancies between the CV claims and the actual LinkedIn profile data.