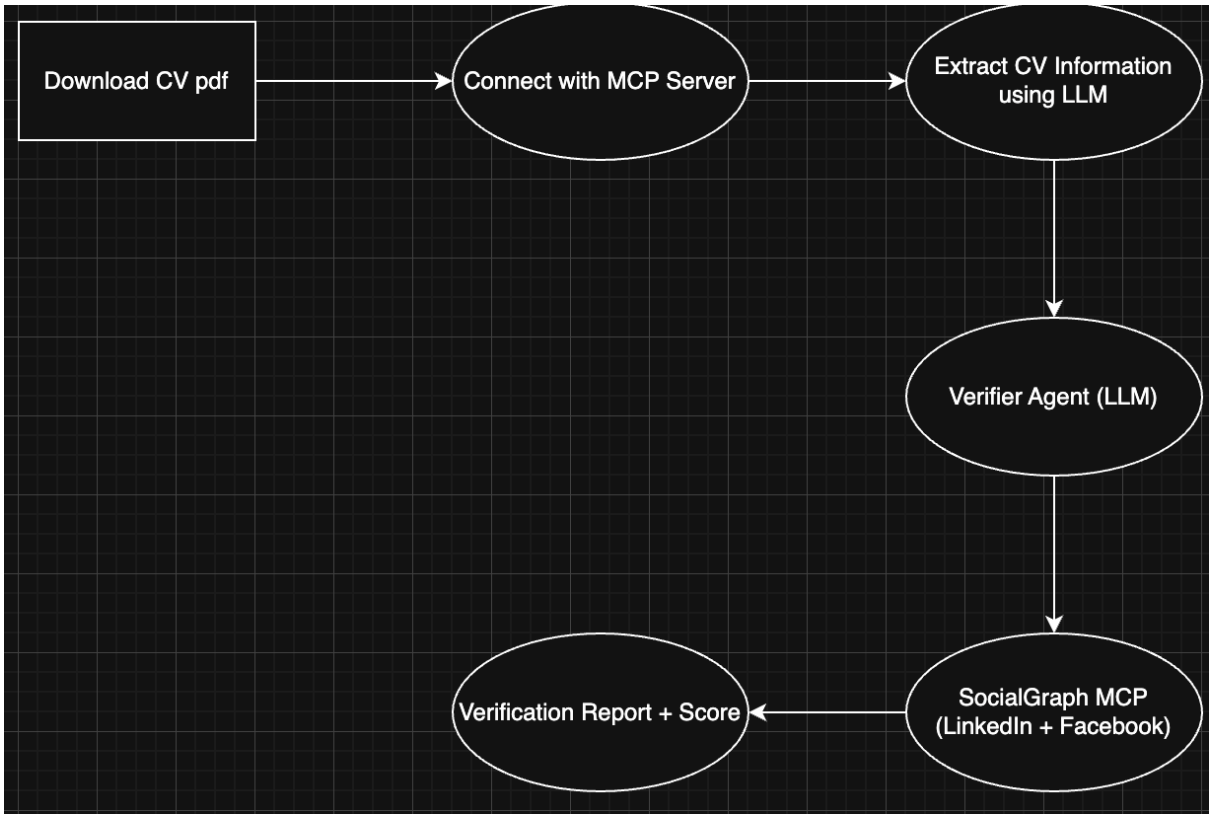
MCP Report

Github: https://github.com/yukic993/FTEC5660HW2_MCP

System Architecture:



Components:

- CV Reader (LLM): Extracts structured fields (name, titles, companies, locations, education, skills) from noisy PDF text.
- Verifier Agent (LLM + Tools): Plans verification, calls MCP tools, compares evidence, assigns scores.
- MCP Tool Layer: SocialGraph tools for LinkedIn and Facebook search and profile retrieval.
- Scoring Module: Deterministic rubric converting discrepancies and identity confidence into a normalized score.

Design Logic

- **LLM-based parsing**: CVs are semi-structured and vary widely in format; an LLM provides robust extraction.
- **LinkedIn as primary source:** Most reliable professional data (roles, companies, education, timelines).
- **Facebook as secondary source:** Supports identity resolution (location, social signals, bio).
- **Agentic loop:** Multi-step planning and tool orchestration is necessary to handle fuzzy matches and ambiguous identities.

- **Score-based decisioning:** Produces a calibrated confidence rather than brittle binary rejection; aligns with evaluator thresholding at 0.5.

Agent Workflow & Tool Usage Strategy

Workflow:

Stage 1 – Plan
The agent plans which fields to verify (name, job, company, location, education, skills) and the order of tool calls.

Stage 2 – Tool Use

- Primary: `search_linkedin_people` → `get_linkedin_profile`
- Secondary: `search_facebook_users` → `get_facebook_profile`
- Fuzzy matching is used when no exact match is found. The agent selects the closest candidate using name similarity, location plausibility, and role/domain overlap.
- The agent never rejects a CV due to missing exact matches and never hallucinates social profiles.

Stage 3 – Evidence Comparison

- Compare CV claims vs. social data for: job titles, companies, locations, education, skills (high-level), and timelines.
- Internal CV inconsistencies (e.g., overlapping roles/PhD, odd locations or degrees) reduce confidence but do not stop verification.

Stage 4 – Scoring
Two outputs are produced:

1. Identity Match Confidence (0–1)
2. Verification Score (0–1) using a deterministic rubric.

Rubic:

Identity Match Confidence Rubric (0–1)

- Name match (max 0.35): exact (0.35), minor variation (0.25), partial (0.15), none (0.00)
- Location consistency (max 0.20): same city (0.20), same country (0.10), regionally plausible (0.05), conflicting (0.00)
- Employer overlap (max 0.25): same company (0.25), same group (0.15), same industry (0.05), none (0.00)
- Education overlap (max 0.15): same university + degree (0.15), same university (0.10), same degree level (0.05), none (0.00)
- Career domain (max 0.05): same (0.05), related (0.03), unrelated (0.00)

Clamped to [0,1]

Interpretation: ≥0.8 very likely same person; 0.6–0.79 likely; 0.4–0.59 ambiguous; <0.4 likely different.

Verification Score Rubric (0–1)

Start from 1.0.

Subtract penalties

- High severity discrepancy: −0.25 each
- Medium severity: −0.15 each
- Low severity: −0.05 each
- Unverified key field among {current_job, company, education}: −0.10 each
- If 0.4 ≤ identity_match_confidence < 0.6: −0.15
- If identity_match_confidence < 0.4: −0.25

Clamp to [0,1].

Decision calibration: Score > 0.5 → VALID; ≤ 0.5 → INVALID (as used by the evaluator).

Sample Verification Results


```
'CV_1.pdf': '```json\n{\n  "identity_match_confidence": 0.6,\n
"verification_score": 0.65,\n
'CV_2.pdf': '```json\n{\n  "identity_match_confidence": 0.75,\n
"verification_score": 0.65,\n
'CV_3.pdf': '```json\n{\n  "identity_match_confidence": 0.4,\n
"verification_score": 0.55,\n
'CV_4.pdf': '```json\n{\n  "identity_match_confidence": 0.35,\n
"verification_score": 0.5,\n
'CV_5.pdf': '```json\n{\n  "identity_match_confidence": 0.75,\n
"verification_score": 0.65,\n
```

```python
def extract_score_from_text(s):
    m = re.search(r'"verification_score"\s*:\s*([0-9.]+)', s)
    if m:
        return float(m.group(1))
    return 0.0

scores = [
    extract_score_from_text(all_results["CV_1.pdf"]),
    extract_score_from_text(all_results["CV_2.pdf"]),
    extract_score_from_text(all_results["CV_3.pdf"]),
    extract_score_from_text(all_results["CV_4.pdf"]),
    extract_score_from_text(all_results["CV_5.pdf"]),
]# Your code should generate this list [0.2, 0.3, 0.4, 0.5, 0.6]
groundtruth = [1, 1, 1, 0, 0] # Do not modify

result = evaluate(scores, groundtruth)
print(result)
```

```
{'decisions': [1, 1, 1, 0, 1], 'correct': 4, 'total': 5, 'final_score': 0.8}
```