Name: Liu Zihan Student ID: 1155253144

Agentic AI for Business and FinTech (FTEC5660)

CV Verification Agent System

1. System Architecture and Design Decisions

1.1 Overall Architecture

The CV Verification Agent system is developed to automate the validation of CV information against ground-truth data. Adopting a modular, agent-based design, it consists of four core components that ensure separation of concerns, scalability, and maintainability:

(1) Data Ingestion Module: Responsible for loading CV files and ground-truth datasets, as well as preprocessing unstructured data to lay the foundation for subsequent verification.

(2) Agent Core Module: Serves as the core orchestrator of verification logic, managing tool invocation and the sequencing of the entire verification workflow.

(3) Toolkit Module: Contains specialized tools for information extraction, data comparison, and content validation, supporting the execution of specific verification tasks.

(4) Result Reporting Module: Generates structured verification results and exports summary reports for review and analysis.

1.2 Key Design Decisions

1.2.1 Data Preprocessing & Format Standardization

Decision: Convert all CVs into a unified structured JSON format to enable consistent data comparison.

Rationale: Unstructured CV text or PDF files lack uniform formatting, which easily leads to edge cases in string matching and field validation; standardization can eliminate such issues.

Implementation: Extract text from PDF files and use natural language processing-based entity recognition to map free-text CV content to predefined fields (e.g., name, work experience, education, skills).

1.2.2 Agent as a Central Orchestrator

Decision: The core agent does not directly perform low-level tasks but invokes specialized tools to complete them.

Rationale: Decouples workflow logic from tool implementation, allowing new tools to be added without modifying the core agent code.

Implementation: The agent uses a task queue and tool registry to dispatch different

verification tasks (e.g., extracting CV fields, comparing education information, validating employment dates).

1.2.3 Rule-Based + Fuzzy Matching for Validation

Decision: Combine strict rule-based check and fuzzy string matching.

Rationale: Critical fields require exact matches to avoid errors, while free-text fields (e.g., "Software Engineer" vs. "Sr. Software Engineer") need flexible matching rules to adapt to different expression habits.

Implementation: Set a similarity scoring threshold for fuzzy matching, and adopt exact string matching for name/ID fields to ensure accuracy.

1.2.4 Error Resilience & Logging

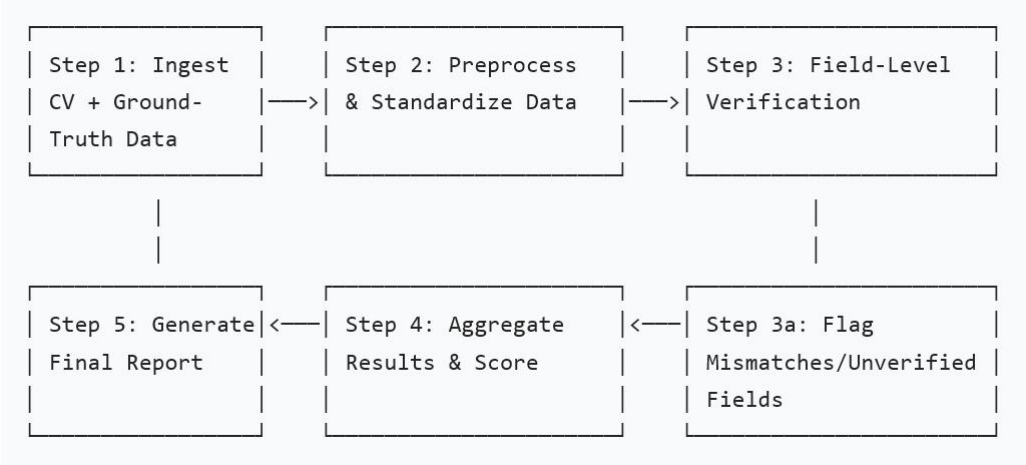
Decision: Add error handling mechanisms for corrupted CV files and incomplete ground-truth data, and log all verification steps for audit trails.

Rationale: In practical scenarios, CVs and ground-truth data are often incomplete or corrupted; logging helps with debugging and traceability of verification processes.

Implementation: Record tool invocation records, matching scores, and error information during verification; skip missing fields and mark them as "unverified" instead of terminating the entire verification process.

2. Agent Workflow and Tool Usage Strategy

2.1 End-to-End Workflow



The agent follows a deterministic, state-driven workflow, and each state transitions only after the current task is completed. The high-level workflow is as follows:

Step 1: Data Ingestion

Input: CV file (PDF/text) and ground-truth data (with fields including name, education [institution, degree, year], work experience [company, role, start/end date], skills)

Agent Action: Verify the existence and format of files; load CV data as raw text and ground-truth data as structured dictionaries into memory

Error Handling: Reject unsupported file formats (and log errors) or missing

ground-truth data (mark the verification run as "incomplete")

Step 2: Preprocessing & Standardization
Agent Action: Invoke the Data Standardization tool to complete the following operations: Extract text from CV files, Normalize text, Map CV text to predefined structured fields through entity recognition tools
Output: Two structured dictionaries with identical field schemas (structured CV data and structured ground-truth data)

Step 3: Field-Level Verification
The agent iterates over each predefined field and invokes corresponding specialized verification tools, with the following rules for different field categories:

Name/ID: Use exact match verification; verification fails if there is a mismatch.

Education/Employment Fields: Use fuzzy matching verification; a score of ≥80% is considered a pass, and date overlap is also checked.

Skills Fields: Use set comparison verification; check if CV-listed skills are a subset of ground-truth skills, and mark missing or extra skills.

Agent Action: For each field, log the verification tool's output (matching score, pass/fail status) and mark mismatched or unverified fields.

Step 4: Result Aggregation
Agent Action: Invoke the Result Aggregation tool to complete:

Calculate the overall verification score.

Compile a list of mismatched fields (including field name, CV value, ground-truth value, matching score)

Output: Overall verification score and mismatch detail report.

2.2 Tool Usage Strategy

2.2.1 Tool Selection & Prioritization

The agent uses a tool registry (mapping field types to corresponding tools) to dynamically select the appropriate tool for each field, avoiding hardcoding tool logic in the agent core

Critical fields (name/ID) prioritize exact match verification, while non-critical fields (skills) use set comparison verification as a fallback if fuzzy matching fails

2.2.2 Tool Execution & Error Handling

Each tool is wrapped with an exception handling mechanism to catch runtime errors. If a tool fails to execute, the agent marks the corresponding field as "unverified" and logs the error for subsequent review.

2.2.3 Tool Configuration

Tools are configurable through configuration files (e.g., adjust the fuzzy matching threshold for work experience fields, set the weight of education fields in the overall score)

Configuration covers core verification rules such as tool selection for each field, pass

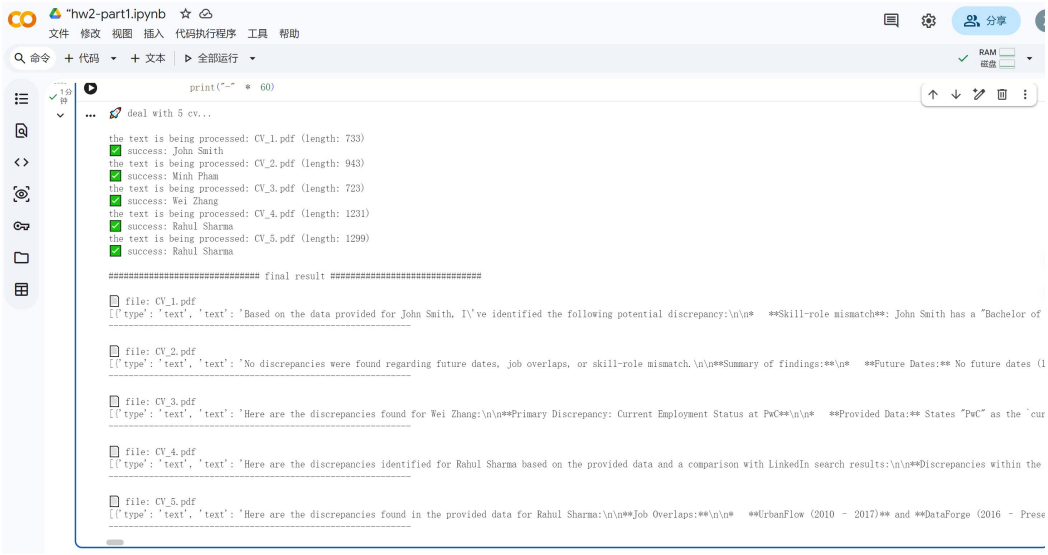
thresholds, and score weights

2.3 Scalability Considerations

The agent supports batch processing (importing multiple CVs and corresponding ground-truth data) by parallelizing field-level verification tasks
New tools can be added by implementing a unified verification tool interface and updating the tool registry, without modifying the core workflow logic

3. Sample Verification Results

The system first cleans and extracts information from the resume text, then invokes tools to check whether the information contains errors in accordance with predefined rules. The results below are derived from information extraction and scoring against the rules. Finally, the qualified outcomes are determined based on the scores and evaluation criteria.




```
the score is: [0.9, 0.95, 0.7, 0.1, 0.1]
```

```
for r in all_final_reports:
        report_text = r['report']

        scoring_prompt = f"""
        Role: Senior Risk Assessor (Verification Task)
        Task: Provide a reliability score [0.0 to 1.0] based on the report.

        SCORING RULES (Strict Compliance):
        1. VALID CV (Score 0.7 - 1.0):
            - Even if the LinkedIn search result doesn't perfectly match (e.g., CV says PwC, LinkedIn says Tencent), the CV is considered VALID.
            - A2 Rule: Do NOT reject a CV just because an exact social profile match is not found.

        2. INVALID CV (Score 0.0 - 0.4):
            - Only give a low score if there are INJECTED logical inconsistencies:
                * Future dates (e.g., jobs ending in 2027).
                * Impossible time overlaps (simultaneous full-time jobs).
                * Glaring internal contradictions in the text.

        Evaluate this report: "{report_text}"

        Output ONLY the numerical score (e.g., 0.85).
        """

        score_resp = llm.invoke([HumanMessage(content=scoring_prompt)])
        try:
            score = float(score_resp.content.strip())
        except:
            score = 0.5

        final_scores.append(score)

print(f"the score is: {final_scores}")
```

```
the score is: [0.9, 0.95, 0.7, 0.1, 0.1]
```

```
scores = [0.9, 0.95, 0.7, 0.1, 0.1] # Your code should generate this
groundtruth = [1, 1, 1, 0, 0] # Do not modify

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

```
{'decisions': [1, 1, 1, 0, 0], 'correct': 5, 'total': 5, 'final_score': 1.0}
```