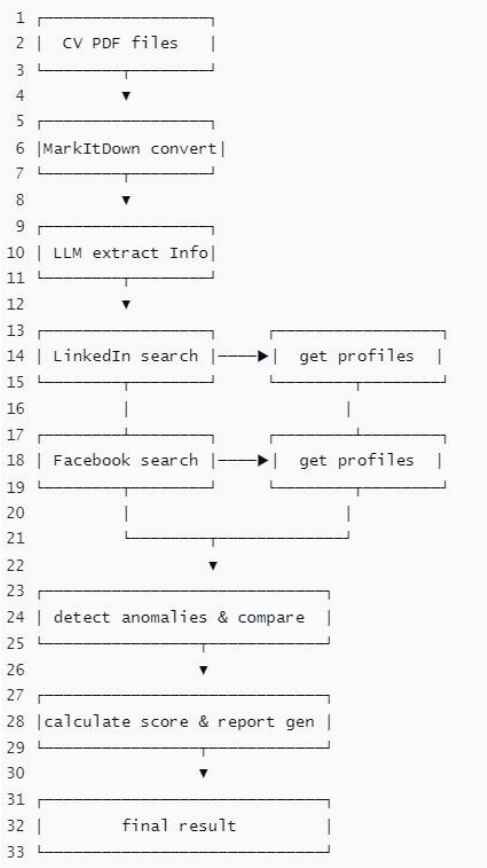
Report - HW2-P1

COURSE: FTEC5660 Agentic AI for Business and FinTech

STUDENT ID/NAME: 1155254873 CHEN,Peng

1. System Description and Agent Workflow.



(1) Input the CV PDF files and convert into markdown format. Here I use the markitdown module and the function which already provided by homework.ipynb .

(2) Use llm to extract the cv's information and output in a structural format. Also I define the verification report's structural class.

```python
class CVInfo(BaseModel):
    name: str
    title: Optional[str] = None
    location: Optional[str] = None
    hometown: Optional[str] = None
    current_company: Optional[str] = None
    education: List[dict] = []
    experience: List[dict] = []
    skills: List[str] = []
```

```python
class VerificationReport(BaseModel):
    cv_name: str
    cv_source: str
    linkedin_match: Optional[dict] = None
    facebook_match: Optional[dict] = None
    discrepancies: List[dict] = []
    verification_score: float = 0.0
    confidence: str = "low"
    match_details: dict = {}
    extracted_info: Optional[CVInfo] = None
```

(3) Search in the Facebook and Linkedin with name extracted. Here is the code for linkedin search. Limit equals 20.

```python
search_tool = next((t for t in mcp_tools if t.name == 'search_linkedin_people'), None)
if not search_tool:
    return None, {}

results = parse_tool_result(await search_tool.ainvoke({'q': cv_info.name, 'limit': 20}))
```

(4) Find the most matched result (here is not the profile, is the search function output). I use keys: name, location, company, to calculate the match score. The algorithm here is the combination of Jaccard and rapidfuzz. Here is part of code.

```
def jaccard_similarity(str1: str, str2: str) -> float:
    if not str1 or not str2:
        return 0.0
    set1, set2 = tokenize(str1), tokenize(str2)
    if not set1 or not set2:
        return 0.0
    return len(set1 & set2) / len(set1 | set2)
```

```
def fuzzy_match_score(str1: str, str2: str) -> float:
    if not str1 or not str2:
        return 0.0
    s1, s2 = str1.lower().strip(), str2.lower().strip()
    scores = [
        fuzz.ratio(s1, s2),
        fuzz.partial_ratio(s1, s2),
        fuzz.token_sort_ratio(s1, s2),
        fuzz.token_set_ratio(s1, s2),
    ]
    return max(scores) / 100.0
```

```
def combined_match(str1: str, str2: str) -> float:
    if not str1 or not str2:
        return 0.0
    return 0.6 * fuzzy_match_score(str1, str2) + 0.4 * jaccard_similarity(str1, str2)
```

(5) After finding the most matched result, I use mcp tool to get the corresponding profiles in Facebook and Linkedin.

```
async def get_profile(mcp_tools, tool_name, id_param, id_val):
    tool = next((t for t in mcp_tools if t.name == tool_name), None)
    if not tool:
        return None
    result = parse_tool_result(await tool.ainvoke({id_param: id_val}))
    if isinstance(result, list) and result:
        result = result[0]
    return result if isinstance(result, dict) and 'id' in result else None
```

And then do more verification with keys such as: experience, company, education. Here is part of code.

```
def compare_profiles(cv_info: CVInfo, linkedin: Optional[dict], facebook: Optional[dict]) -> List[dict]:
    issues = []

    if linkedin:
        li_exp = linkedin.get('experience', [])
        li_current = [e for e in li_exp if isinstance(e, dict) and e.get('is_current')]
```

```
    cv_company = cv_info.current_company
    if cv_info.experience:
        cv_company = cv_info.experience[0].get('company', cv_company)

    if cv_company and li_current:
        li_company = li_current[0].get('company', '')
        if combined_match(cv_company, li_company) < 0.4:
            issues.append({
                'type': 'company_mismatch_linkedin',
                'severity': 'low',
                'description': f"CV: {cv_company} vs LinkedIn: {li_company}"
            })
```

If there are some discrepancies, I will append a dict with specific key - “severity” which will cause reducing points in the following processes.

(6) Detect time anomalies. This function is aimed to find the discrepancies inside the original CV.

```
def detect_time_anomalies(cv_info: CVInfo) -> List[dict]:
    current_year = datetime.datetime.now().year
    anomalies = []
```

```
for exp in cv_info.experience:
    start = exp.get('start_year')
    end = exp.get('end_year')

    if start:
        try:
            s = int(start) if str(start).isdigit() else None
            if s and s > current_year:
                anomalies.append({
                    'type': 'future_start_date',
                    'severity': 'critical',
                    'description': f"Work starts in future: {start}"
                })
        except:
            pass
```

If there are discrepancies, the final score will be reduced. A dict with key - “severity” will be appended just like the previous verification processes.

(7) Calculate the score. I set the original score is 0.5 , if the profile is matching well, then add the points, if there are discrepancies, then reduce the points.

```
def calculate_score(discrepancies: List[dict], li_details: dict, fb_details: dict) -> float:
    score = 0.5
```

```
if li_details:
    li_total = li_details.get('total', 0)
    if li_total >= 30:
        score += 0.20
    elif li_total >= 25:
        score += 0.15
    elif li_total >= 20:
        score += 0.10
    elif li_total >= 15:
        score += 0.05
```

```
critical_count = sum(1 for d in discrepancies if d.get('severity') == 'critical')
high_count = sum(1 for d in discrepancies if d.get('severity') == 'high')
medium_count = sum(1 for d in discrepancies if d.get('severity') == 'medium')
low_count = sum(1 for d in discrepancies if d.get('severity') == 'low')

score -= critical_count * 0.20
score -= high_count * 0.15
score -= medium_count * 0.1
score -= low_count * 0.05
```

(8) Define the function that verify a single cv and generate the corresponding verification report.

```
async def verify_single_cv(llm, mcp_tools, cv_text: str, cv_source: str) -> VerificationReport:
    cv_info = await extract_cv_info(llm, cv_text)
    print(f"  Extracted: {cv_info.name}")
```

And then define the function that can handle multiple cvs (list of markdown).

```
async def verify_cv_list(llm, mcp_tools, cv_texts: List[str], sources: List[str] = None)
    if sources is None:
        sources = [f"input_{i+1}" for i in range(len(cv_texts))]
```

```
    scores, reports = [], []

    for cv_text, source in zip[tuple[str, str]](cv_texts, sources):
        print(f"\n{'='*60}\nProcessing: {source}\n{'='*60}")
        report = await verify_single_cv(llm, mcp_tools, cv_text, source)
        reports.append(report)
        scores.append(report.verification_score)
        print(f"  Score: {report.verification_score:.2f} ({report.confidence})")
        for d in report.discrepancies:
            print(f"    ! {d['type']}: {d.get('description', '')}")

    print(f"\n{'='*60}\nSUMMARY\n{'='*60}")
    print(f"Scores: {[round(s, 2) for s in scores]}")
    return scores, reports
```

(9) How to start the agent. I define a class called “CVVerificationAgent”, this class will initial with llm and mcp tools, also it has functions that can verify the pdfs or markdowns. When I test the code, I create an instance, and use function - verify_markdown_list to see the result.

```
class CVVerificationAgent:
    def __init__(self):
        from dotenv import load_dotenv
        load_dotenv()
        self.api_key = os.getenv("ARK_API_KEY")
        self.api_base = os.getenv("ARK_API_URL")
        self.model_name = os.getenv("ARK_API_MODEL", "deepseek-v3-2-251201")
        self.llm = None
        self.mcp_tools = None
        self._initialized = False
```

```python
async def initialize(self):
    from langchain_openai import ChatOpenAI
    self.llm = ChatOpenAI(model=self.model_name, api_key=self.api_key, base_url=self.api_base, temperature=0)
    client = MultiServerMCPClient({
        "social_graph": {"transport": "http", "url": "https://ftec5660.ngrok.app/mcp",
                         "headers": {"ngrok-skip-browser-warning": "true"}}
    })
    self.mcp_tools = await client.get_tools()
    self._initialized = True
    print("Agent initialized")
```

2. Results and Valuation

Create a instance, call the function the test.

```python
# create the CVVerificationAgent instance
agent = CVVerificationAgent()

# Here is the interface to verify the markdown texts
scores, reports = await agent.verify_markdown_list(markdown_texts, pdf_names)
```

Evaluation process.

```python
print(f"\nFinal Scores: {[round(s, 2) for s in scores]}")
result = evaluate(scores, [1, 1, 1, 0, 0])
print(f"Evaluation: {result}")
return scores, reports
```

Final result and verification reports:

```
============================================================
Processing: CV_1.pdf
============================================================
  Extracted: John Smith
  LinkedIn: John Smith (score: 29.8)
  Facebook: John Smith (score: 28.0)
  Score: 0.55 (medium)
    ! company_mismatch_linkedin: CV: ByteDance vs LinkedIn: Manulife
    ! education_mismatch_linkedin: CV school: McGill University not found in LinkedIr
    ! company_mismatch_facebook: CV: ByteDance vs Facebook: Traveloka

============================================================
Processing: CV_2.pdf
============================================================
  Extracted: Minh Pham
  LinkedIn: Minh Pham (score: 30.0)
  Facebook: Minh Pham (score: 27.8)
  Score: 0.75 (high)
    ! company_mismatch_facebook: CV: BCG vs Facebook: Manulife

============================================================
Processing: CV_3.pdf
============================================================
  Extracted: Wei Zhang
  LinkedIn: Wei Zhang (score: 31.0)
  Facebook: Wei Zhang (score: 27.8)
  Score: 0.85 (high)
```

```
============================================================
Processing: CV_4.pdf
============================================================
  Extracted: Rahul Sharma
  LinkedIn: Rahul Sharma (score: 28.1)
  Facebook: Rahul Sharma (score: 28.0)
  Score: 0.25 (low)
    ! future_end_date: Work ends in future: 2027
    ! overlapping_experience: Overlapping: Microsoft (2021-2027) and StartupXYZ (2020-2023)
    ! company_mismatch_linkedin: CV: Microsoft vs LinkedIn: Meta
    ! company_mismatch_facebook: CV: Microsoft vs Facebook: Shopee

============================================================
Processing: CV_5.pdf
============================================================
  Extracted: Rahul Sharma
  LinkedIn: Rahul Sharma (score: 29.1)
  Facebook: Rahul Sharma (score: 28.0)
  Score: 0.40 (medium)
    ! overlapping_experience: Overlapping: StartupXYZ (2019-2021) and DataForge (2016-Present)
    ! overlapping_experience: Overlapping: DataForge (2016-Present) and UrbanFlow (2010-2017)

============================================================
SUMMARY
============================================================
Scores: [0.55, 0.75, 0.85, 0.25, 0.4]

Final Scores: [0.55, 0.75, 0.85, 0.25, 0.4]
Evaluation: {'decisions': [1, 1, 1, 0, 0], 'correct': 5, 'total': 5, 'final_score': 1.0}
```