HW2 part1

Chen Jiaxin 1155246854

Step1 Install Required Packages

```
!pip install requests PyPDF2 gdown
!pip install markitdown[pdf]
!pip install langchain_mcp_adapters langchain_google_genai langchain-openai
```

显示隐藏的输出项

Step2 Import Libraries

```
import asyncio
import json
import re
import os
from google.colab import userdata
from langchain_openai import ChatOpenAI
from langchain_core.messages import HumanMessage
from langchain_mcp_adapters.client import MultiServerMCPClient
from markitdown import MarkItDown
import gdown
```

Step3 Set API Keys and MCP Client

```
DEEPSEEK_API_KEY = userdata.get('DEEPSEEK_API_KEY')
llm = ChatOpenAI(
    model="deepseek-chat",
    api_key=DEEPSEEK_API_KEY,
    base_url="https://api.deepseek.com/v1",
    temperature=0
)
```

Step4 Download CV Files

```
folder_id = "1adYKq7gSSczFP3iikfA8Er-HSZP6VM7D"
folder_url = f"https://drive.google.com/drive/folders/{folder_id}"
output_dir = "downloaded_cvs"
os.makedirs(output_dir, exist_ok=True)
gdown.download_folder(
    url=folder_url,
    output=output_dir,
    quiet=False,
    use_cookies=False
)
```

显示隐藏的输出项

Step5 Initialize MCP Client and Tools

```
client = MultiServerMCPClient({
    "social_graph": {
        "transport": "http",
        "url": "https://ftec5660.ngrok.app/mcp",
        "headers": {"ngrok-skip-browser-warning": "true"}
    }
})
mcp_tools = await client.get_tools()
FACEBOOK_SEARCH = 0
FACEBOOK_PROFILE = 1
LINKEDIN_SEARCH = 3
LINKEDIN_PROFILE = 4
```

Step6 Extract Information from CV

We extract the key information from the CV, such as name, city, education, and experience, to facilitate comparison across multiple platforms later.

```
def extract_cv_info(cv_text):
    prompt = f"""
```

```
        Extract structured information from this CV.
        Return ONLY valid JSON.

        Format:
        {{
            "name": "",
            "city": "",
            "education": [],
            "experience": []
        }}

        CV:
        {cv_text}
        """
        response = llm.invoke([HumanMessage(content=prompt)])
        content = response.content.strip()

        try:
            json_match = re.search(r"\{.*\}", content, re.DOTALL)
            if json_match:
                return json.loads(json_match.group(0))
        except:
            pass

        return {"name": "", "city": "", "education": [], "experience": []}
```

Step7 LinkedIn Search (LLM Selects Best Match)

In this process, first search for the 10 candidates that best match the relevant information based on keywords, and then let the LLM determine which one best meets the requirements.

```
    async def search_linkedin(cv_info):
        try:
            city = cv_info.get("city", "")
            if "," in city:
                city = city.split(",")[0].strip()
            search_results = await mcp_tools[LINKEDIN_SEARCH].ainvoke({
                "q": cv_info.get("name", ""),
                "location": city,
                "limit": 10,
                "fuzzy": True
            })

            if not search_results:
                return None

            linkedin_people = json.loads(search_results[0]["text"])
            if not linkedin_people:
                return None

            selection_prompt = f"""
            CV:
            {json.dumps(cv_info, indent=2)}

            Candidates:
            {json.dumps(linkedin_people, indent=2)}

            Select the BEST matching profile.
            Return ONLY the profile id.
            If none match, return NONE.
            """
            response = llm.invoke([HumanMessage(content=selection_prompt)])
            selected_id = response.content.strip()

            if selected_id == "NONE":
                return None

            selected_id = int(re.search(r"\d+", selected_id).group())

            profile_result = await mcp_tools[LINKEDIN_PROFILE].ainvoke({
                "person_id": selected_id
            })

            return json.loads(profile_result[0]["text"])

        except Exception as e:
            print("LinkedIn error:", e)
            return None
```

Step8 Facebook Search (LLM Selects Best Match)

The same as the step7.first search for the 10 candidates from Facebook that best match the relevant information based on keywords, and then let the LLM determine which one best meets the requirements.

```
async def search_facebook(cv_info):
    try:
        search_results = await mcp_tools[FACEBOOK_SEARCH].ainvoke({
            "q": cv_info.get("name", ""),
            "limit": 10,
            "fuzzy": True
        })

        if not search_results:
            return None

        fb_users = json.loads(search_results[0]["text"])
        if not fb_users:
            return None

        selection_prompt = f"""
        CV:
        {json.dumps(cv_info, indent=2)}

        Facebook Candidates:
        {json.dumps(fb_users, indent=2)}

        Select BEST match.
        Return ONLY user_id.
        If none match, return NONE.
        """

        response = llm.invoke([HumanMessage(content=selection_prompt)])
        selected_id = response.content.strip()

        if selected_id == "NONE":
            return None

        selected_id = int(re.search(r"\d+", selected_id).group())

        profile_result = await mcp_tools[FACEBOOK_PROFILE].ainvoke({
            "user_id": selected_id
        })
        return json.loads(profile_result[0]["text"])
    except Exception as e:
        print("Facebook error:", e)
        return None
```

Step9 LLM Verification Decision

Based on all the information obtained earlier, the LLM evaluates the identity consistency between the CV and its corresponding LinkedIn and Facebook profiles. First, the model is instructed to score and provide explanations across four dimensions: name, education background, work experience, and geographic location (each dimension scored 0–1). The JSON results returned by the model are then parsed. A weighted average is calculated based on my designed weights ("name_score": 0.25, "education_score": 0.35, "experience_score": 0.25, "location_score": 0.15) across the four dimensions to derive a final comprehensive matching score (constrained between 0–1). If parsing fails, a default value of 0 is returned.

```
def llm_verification_decision(cv_info, linkedin_profile, facebook_profile):
    prompt = f"""
You are a strict KYC identity verification system.

Compare the CV with LinkedIn and Facebook profiles.

CV:
{json.dumps(cv_info, indent=2)}

LinkedIn:
{json.dumps(linkedin_profile, indent=2) if linkedin_profile else "Not Found"}

Facebook:
{json.dumps(facebook_profile, indent=2) if facebook_profile else "Not Found"}
----------------------------------
TASK:

Evaluate identity consistency across:

1. Name consistency
2. Education consistency
3. Experience / Company consistency
4. City / Location consistency
```

```
Scoring Rules:
- 1.0 = fully consistent
- 0.0 = completely inconsistent
- Intermediate values allowed
- Missing LinkedIn should NOT be penalized
- If LinkedIn not found but Facebook matches name and location,
  treat as weak positive evidence
- Facebook job mismatch alone should not cause very low experience_score
- If company completely mismatches → experience_score must be low
- If education totally different → education_score must be low
- If names differ significantly → name_score must be low
- Missing Facebook is NOT heavily penalized

----------------------------------

Return ONLY valid JSON in this format:

{{
  "name_score": 0-1,
  "education_score": 0-1,
  "experience_score": 0-1,
  "location_score": 0-1,
  "reason": "short explanation"
}}
"""

    response = llm.invoke([HumanMessage(content=prompt)])
    content = response.content.strip()

    try:
        json_match = re.search(r"\{.*\}", content, re.DOTALL)
        if not json_match:
            return 0

        result = json.loads(json_match.group(0))

        weights = {
            "name_score": 0.25,
            "education_score": 0.35,
            "experience_score": 0.25,
            "location_score": 0.15
        }

        final_score = 0
        for key, weight in weights.items():
            score_value = result.get(key, 0)
            try:
                score_value = float(score_value)
            except:
                score_value = 0
            final_score += score_value * weight

        final_score = max(0.0, min(1.0, final_score))

        return final_score

    except Exception as e:
        print("Score parse error:", e)
        return 0
```

Step10 Main Processing Function

The main purpose of this step is to batch process all PDF resumes. It iterates through all PDF files in the specified directory, reads and parses each one into text, extracts structured CV information, then searches for corresponding profiles on LinkedIn and Facebook respectively, prints debugging information, and finally calls the previously defined LLM scoring function to calculate identity matching scores, storing each resume's final score in a list for return.If no relevant information can be found on either platform, select the most similar ones and re-evaluate.

```
async def process_all_cvs():
    md = MarkItDown(enable_plugins=False)

    pdf_files = sorted(
        [f for f in os.listdir(output_dir) if f.lower().endswith(".pdf")],
        key=lambda x: int("".join(filter(str.isdigit, x)) or "0")
    )
    scores = []
    for pdf_name in pdf_files:
        print(f"\nProcessing {pdf_name}")
        pdf_path = os.path.join(output_dir, pdf_name)
```

```
        result = md.convert(pdf_path)
        cv_info = extract_cv_info(result.text_content)
        linkedin_profile = await search_linkedin(cv_info)
        facebook_profile = await search_facebook(cv_info)
        print("LinkedIn Profile Found:", linkedin_profile is not None)
        print("Facebook Profile Found:", facebook_profile is not None)
        if linkedin_profile is None and facebook_profile is None:
            print("Both platforms failed. Re-running search to force best match...")

            linkedin_profile = await search_linkedin(cv_info)
            facebook_profile = await search_facebook(cv_info)

        decision = llm_verification_decision(
            cv_info,
            linkedin_profile,
            facebook_profile
        )

        scores.append(decision)
        print("Decision:", decision)
    return scores
scores = await process_all_cvs()
print("\nFINAL SCORES:", scores)
```

```
Processing CV_1.pdf
LinkedIn Profile Found: False
Facebook Profile Found: True
Decision: 0.5549999999999999

Processing CV_2.pdf
LinkedIn Profile Found: True
Facebook Profile Found: True
Decision: 0.875

Processing CV_3.pdf
LinkedIn Profile Found: False
Facebook Profile Found: True
Decision: 0.73

Processing CV_4.pdf
LinkedIn Profile Found: False
Facebook Profile Found: False
Both platforms failed. Re-running search to force best match...
Decision: 0.0

Processing CV_5.pdf
LinkedIn Profile Found: True
Facebook Profile Found: True
Decision: 0.42500000000000004

FINAL SCORES: [0.5549999999999999, 0.875, 0.73, 0.0, 0.42500000000000004]
```

Step11 Evaluation Code

```
def evaluate(scores, groundtruth, threshold=0.5):

    assert len(scores) == 5
    assert len(groundtruth) == 5

    correct = 0
    decisions = []

    for s, gt in zip(scores, groundtruth):
        pred = 1 if s > threshold else 0
        decisions.append(pred)
        if pred == gt:
            correct += 1
    final_score = correct / len(scores)

    return {
        "decisions": decisions,
        "correct": correct,
        "total": len(scores),
        "final_score": final_score
    }
groundtruth = [1, 1, 1, 0, 0]
result = evaluate(scores, groundtruth)
print("Evaluation Result:")
print(result)
```

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