REPORT

소프트웨어공학 6팀

(월 4-6교시)

20210868 장세준
20200679 하도윤
20210866 임원석
20210881 하승훈
20210988 설동화
20220280 한창하

<목차>

1. 개요

2. 모듈별 구조 및 역할

3. 실행

4. 업무 분장

1. 개요

▶주제 : 사진 기반 음식 레시피 시스템

모든 사람은 음식을 섭취함으로 에너지를 얻고 살아갑니다. 그러나 음식을 조리하는데 여러 사정으로 인해 어려움을 겪는 사람이 있습니다. 음식을 조리하는 것에 쉽게 접근하고자 저희 6팀은 집에 있는 재료로 음식을 추천하고 레시피를 알려주는 시스템을 개발했습니다.

▶기능

-음식 재료가 보이게 사진을 찍어 시스템에 찍은 사진 경로를 입력

-시스템이 사진을 확인하고 어떤 재료인지 분석

-분석 완료 후 재료로 조리할 수 있는 음식 추천

-추천한 음식의 레시피 및 조리법 출력

-사용자의 질문에 응답 후 종료

2. 모듈별 구조 및 역할

▶main.py

기능 : 프로그램의 진입점
run_cli() 호출을 통해 CLI 기반 인터페이스를 실행한다.
종속 모듈: vlm_processor, recipe_ai, qa_bot, terminal_ui

```
from vlm_processor import get_ingredients
from recipe_ai import generate_recipe
from qa_bot import answer
from terminal_ui import run_cli


def main():
    # CLI 모드로 실행
    run_cli()


if __name__ == "__main__":
    main()
```

▶terminal_ui.py

기능: 사용자와의 상호작용을 위한 CLI UI 제공한다.
순서대로 다음 기능 수행 :
사용자로부터 냉장고 이미지 경로 입력한다.
이미지 → 식재료 추출 (get_ingredients)
식재료 → 레시피 생성 (generate_recipe)
레시피에 대해 사용자 질문 응답 (answer)
핵심 흐름:입력(이미지) → 인식 → 레시피 생성 → 질문 → 응답

```
from vlm_processor import get_ingredients
from recipe_ai import generate_recipe
from qa_bot import answer

def run_cli():
    print(" 냉장고 AI 셰프에 오신 것을 환영합니다!")

    # 1. 이미지 경로 입력
    img_path = input(" 분석할 냉장고 이미지 경로를 입력하세요: ").strip()
```

```
    # 2. 식재료 인식
    print("₩n 식재료 인식 중입니다...")
    try:
        ingredients = get_ingredients(img_path)
        if not ingredients:
            print(" 재료를 인식하지 못했습니다. 이미지를 다시 확인해주세요.")
            return
        print(" 인식된 재료:", ingredients)
    except Exception as e:
        print(" 재료 인식 실패:", e)
        return

    # 3. 레시피 생성
    print("₩n 레시피 생성 중입니다... 잠시만 기다려주세요.")
    try:
        recipe = generate_recipe(ingredients)
        print("₩n 생성된 레시피:₩n")
        print(recipe)
    except Exception as e:
        print(" 레시피 생성 실패:", e)
        return

    # 4. 사용자 질문 → LLM 답변
    print("₩n 요리에 대해 궁금한 점을 입력해보세요 (엔터만 입력 시 종료)")
    while True:
        question = input("₩n 질문: ").strip()
        if question == "":
            print(" 종료합니다. 즐거운 요리 되세요!")
            break
        try:
            reply = answer(question, recipe)
            print(" 답변:", reply)
        except Exception as e:
            print(" 답변 실패:", e)
    # 5. 파일 단독 실행 코드

if __name__ == "__main__":
    run_cli()
```

▶vlm_processor.py

기능 : 이미지 기반 식재료 인식 (Vision Language Model 사용)
Ollama 서버의 Gemma 3 멀티모달 모델에 이미지와 프롬프트를 보내어 식재료를 추출한다.
정규표현식을 사용해 모델 응답을 정제 → 문자열 리스트 반환한다.
오류 상황(모델 미설치, 서버 미실행, 형식 오류 등)에 대한 예외 처리 내장한다.

```
import sys # 시스템 관련 기능 (표준 에러 출력 등)
import json # JSON 데이터 처리 (Ollama API 응답 파싱용)
import re # 정규 표현식 (모델 응답 텍스트 파싱용)
from pathlib import Path # 파일 경로를 객체로 다루기 위함 (경로 존재 여부 확인 등)

# Ollama 파이썬 클라이언트 라이브러리 임포트
# 이 라이브러리는 HTTP 요청 및 Base64 인코딩을 내부적으로 처리하여
# Ollama 서버와 편리하게 통신할 수 있도록 돕습니다.
from ollama import chat

# 스크립트 시작을 알리는 로그 출력
print("--- Script execution started: vlm_processor_test.py ---")

# --- VLM 모듈의 핵심 함수 ---
# 이 함수는 VLM 담당자가 구현하여 다른 모듈에서 호출할 최종 모듈의 인터페이스입니다.
# 즉, 이 프로젝트에서 다른 팀원(예: 레시피 생성 담당자)이 이 함수를 호출하여
# 이미지로부터 식재료 목록을 얻게 됩니다.
def get_ingredients(image_path: str) -> list[str]:
    """
    이미지 파일 경로를 입력받아 Ollama의 Gemma 3 모델을 사용하여
    이미지 속 식재료 목록을 텍스트로 추출하고, 이를 파싱하여 문자열 리스트로 반환합니다.

    Args:
        image_path (str): 분석할 이미지 파일의 절대 또는 상대 경로.

    Returns:
        list[str]: 이미지에서 인식된 식재료 이름들의 리스트.
                   오류 발생 시 빈 리스트를 반환합니다.
    """
    # 1. 이미지 파일 존재 여부 확인 로직
    # 제공된 image_path가 실제 파일인지 확인합니다.
    if not Path(image_path).is_file():
        print(f"Error: Image file not found at {image_path}", file=sys.stderr)
        return [] # 파일이 없으면 빈 리스트 반환 후 함수 종료
```

```
    try:
        # 2. Ollama chat 함수에 전달할 메시지 구성
    # Gemma 3와 같은 멀티모달 모델은 'images' 필드를 통해 이미지 입력을 받습니다.
    # 'content' 필드에는 이미지에 대한 질문(프롬프트)을 작성합니다.
# 프롬프트는 모델이 원하는 형태로 응답하도록 유도하는 것이 중요합니다 (예: 쉼표로 구분).
        vlm_prompt_text = "What ingredients are shown in this picture? List them only, separated by commas, like 'onion, potato, carrot'."

        messages = [
            {"role": "user",
             "content": vlm_prompt_text,
             "images": [image_path] # 핵심: 이미지 파일 경로를 'images' 리스트에 담아 전달
                                    # Ollama 라이브러리가 이 경로의 이미지를 읽어 Base64로 자동 인코딩합니다.
            }
        ]

        print(f"₩n[VLM Module] Sending request to Ollama with image: {image_path}...")

        # 3. Ollama chat API 호출
        # chat() 함수를 사용하여 Ollama 서버와 통신합니다.
        # - model: 사용할 LLM/VLM 모델의 이름과 태그 (예: "gemma3:4b").
        # - messages: 대화 이력을 포함하는 메시지 리스트.
        # - options: 모델의 응답 방식 제어 (예: temperature는 창의성, 0.1은 비교적 일관된 답변 유도).
        # - stream=False: 응답을 실시간 스트리밍하지 않고, 모든 응답이 완료된 후 한 번에 받습니다.
        # (참고: ollama.chat() 함수는 timeout 인자를 직접 받지 않으므로 제거되어 있습니다.)
        response = chat(model="gemma3:4b", messages=messages, options={"temperature": 0.1}, stream=False)

        # Ollama 모델의 응답에서 실제 텍스트 내용만 추출하여 공백 제거
        raw_vlm_response = response['message']['content'].strip()
        print(f"[VLM Module] Raw Gemma 3 VLM Response: {raw_vlm_response}")

        # 4. 응답 텍스트에서 식재료만 파싱하는 로직
        # 이 부분은 Gemma 3의 실제 응답 패턴에 따라 가장 '효과적'이고 '견고'하게
```

```
        # 식재료만 추출하도록 다듬어야 하는 VLM 담당자의 주요 과업입니다.
        parsed_ingredients = []
        if raw_vlm_response:
         # Gemma 3의 응답에서 불필요한 서두 문구들을 정규 표현식을 사용하여 제거
         # 예: "The ingredients are", "I can see", "This image appears to contain" 등
            cleaned_response = re.sub(
                r"^(The ingredients are|The image shows|I can see|Ingredients:|This image appears to contain)₩s*",
                "",
                raw_vlm_response,
                flags=re.IGNORECASE # 대소문자 구분 없이 매칭
            ).strip()

            # 응답의 마지막에 마침표가 있다면 제거 (깔끔한 파싱을 위함)
            if cleaned_response.endswith('.'):
                cleaned_response = cleaned_response[:-1]

           # 쉼표(,)를 기준으로 문자열을 분리하고, 각 요소를 공백 제거 후 리스트에 추가
            # 빈 문자열은 제외합니다.
            parts = [part.strip() for part in cleaned_response.split(',') if part.strip()]
            parsed_ingredients = parts

        return parsed_ingredients

    # 예외 처리: Ollama 서버 통신 또는 모델 처리 중 발생할 수 있는 오류를 잡습니다.
    except Exception as e:
        print(f"₩n[VLM Module Error] An error occurred during Ollama chat: {e}", file=sys.stderr)

        # 오류 메시지 내용을 기반으로 사용자에게 더 구체적인 힌트를 제공합니다.
        error_message_lower = str(e).lower()
        if "model not found" in error_message_lower or "404" in error_message_lower:
            print("   Hint: Model 'gemma3:4b' not found on Ollama server. Check 'ollama list' or 'ollama pull gemma3:4b'.", file=sys.stderr)
        elif "timeout" in error_message_lower: # 라이브러리 내부에서 발생하는 타임아웃 오류
            print("   Hint: The request timed out. Model might be processing slowly due to large image or low resources. Check Ollama server logs.", file=sys.stderr)
        elif "connection refused" in error_message_lower or "connectionerror" in error_message_lower:
```

```
            print("  Hint: Could not connect to Ollama server. Is 'ollama serve' running? (It should be running in a separate terminal.)", file=sys.stderr)
        elif "jsondecodeerror" in error_message_lower:
            print("  Hint: Received an unparseable response from Ollama. Check Ollama server logs for errors or unexpected output.", file=sys.stderr)

        return [] # 오류 발생 시 빈 리스트 반환

# --- 메인 실행 블록 ---
# 이 부분은 VLM 모듈의 기능을 테스트하고 시연하기 위한 코드입니다.
# 실제 통합 시에는 '통합 & 발표' 담당자가 이 함수(get_ingredients)를 호출하게 됩니다.
if __name__ == "__main__":
    # 테스트할 이미지 경로들을 리스트로 정의합니다.
    # 이 경로는 실제 이미지 파일의 위치와 일치해야 합니다.
    # 테스트할 이미지 경로가 다를 수 있으므로, 자신의 환경에 맞게 수정해야 합니다.
    test_image_paths = [

r"C:₩Users₩hadyo₩Downloads₩swg-main₩swg-main₩sample_image₩sample_image.png",
# 첫 번째 샘플 이미지 경로

r"C:₩Users₩hadyo₩Downloads₩swg-main₩swg-main₩sample_image₩sample_image2.png",
# 두 번째 샘플 이미지 경로

r"C:₩Users₩hadyo₩Downloads₩swg-main₩swg-main₩sample_image₩sample_image3.png"
 # 세 번째 샘플 이미지 경로
    ]

    print(f"--- Running VLM Module Test for Multiple Images ---")

    # 각 이미지 경로에 대해 반복하여 VLM 모듈 테스트를 실행합니다.
    for i, image_path in enumerate(test_image_paths):
        print(f"₩n===== Testing Image {i+1}: {image_path} =====")

        # VLM 모듈의 핵심 함수 호출
        recognized_ingredients = get_ingredients(image_path)

        # 테스트 결과 출력
        if recognized_ingredients:
            print(f"₩n[VLM Module Result] Recognized Ingredients: {recognized_ingredients}")
            print(f"Success! The VLM module successfully extracted ingredients for
```

```
{Path(image_path).name}.")
        else:
            print(f"₩n[VLM Module Result] No ingredients recognized or an error occurred for {Path(image_path).name}.")
            print(f"Please check the image, Ollama server status, and VLM parsing logic if this was unexpected.")

        print(f"===== Test for Image {i+1} Finished =====")
```

▶recipe_ai.py

기능 : 식재료 기반 자동 레시피 생성
프롬프트에 따라 구조화된 레시피 포맷을 LLM에게 요청하여 생성한다.
Gemma 3 LLM을 사용하며, 결과는 문자열로 반환된다.

```
from ollama import chat

def generate_recipe(ingredients: list[str]) -> str:
    """
    주어진 식재료 목록을 바탕으로 요리 레시피를 생성합니다.

    Args:
        ingredients (list[str]): VLM이 인식한 식재료 이름 목록

    Returns:
        str: 생성된 레시피 텍스트
    """
    # 식재료 목록을 문자열로 변환
    ingredients_text = ", ".join(ingredients)

    # 레시피 생성을 위한 프롬프트 구성
    prompt = f"""
    냉장고에 있는 재료: {ingredients_text}

    위 재료들로 만들 수 있는 요리를 한 가지 추천해주세요.
    결과는 반드시 다음 형식으로 보여주세요:
    try:
        # Ollama API를 통해 레시피 생성 요청
        response = chat(
            model="gemma3:4b",
            messages=[{
```

```
                "role": "user",
                "content": prompt
            }],
            options={"temperature": 0.7}  # 약간의 창의성 허용
        )

        # 응답에서 레시피 텍스트 추출
        recipe_text = response['message']['content'].strip()
        return recipe_text

    except Exception as e:
        print(f"레시피 생성 중 오류 발생: {e}")
        return f"레시피를 생성하는 중 오류가 발생했습니다: {str(e)}"
```

▶qa_bot.py

기능 : 레시피 기반 질의응답 처리
사용자의 요리 관련 질문에 대해 LLM(Gemma 3)으로부터 답변 생성한다.
레시피 내용이 context로 함께 전달되며, 답변은 간결하게 제한된다.
Temperature를 낮게 설정해 일관되고 정확한 응답 유도한다.

```
from ollama import chat

def answer(question: str, recipe_context: str = "") -> str:
    """
    사용자의 요리 관련 질문에 답변합니다.

    Args:
        question (str): 사용자의 질문
        recipe_context (str, optional): 레시피 컨텍스트. 기본값은 빈 문자열

    Returns:
        str: 질문에 대한 답변
    """
    # 시스템 프롬프트 설정
    system_prompt = """당신은 유저의 요리 조리법 관련 질문에 1~2 문장으로
    간결하고 정확하게 답변하는 전문 요리 도우미 QA 봇입니다."""

    # 메시지 구성
    messages = [
        {"role": "system", "content": system_prompt}
```

```
]

# 레시피 컨텍스트가 있다면 추가
if recipe_context:
    messages.append({
        "role": "system",
        "content": f"Context:₩n{recipe_context}"
    })

# 사용자 질문 추가
messages.append({
    "role": "user",
    "content": question
})

try:
    # Ollama API를 통해 답변 생성 요청
    response = chat(
        model="gemma3:4b",
        messages=messages,
        options={"temperature": 0.2}  # 낮은 temperature로 일관된 답변 유도
    )

    # 응답에서 답변 텍스트 추출
    answer_text = response['message']['content'].strip()
    return answer_text

except Exception as e:
    print(f"답변 생성 중 오류 발생: {e}")
    return f"답변을 생성하는 중 오류가 발생했습니다: {str(e)}"
```

3. 실행

▶실행 시 주의사항

-Ollama 서버 실행 필요 (ollama serve)
-gemma3:4b 모델 사전 다운로드 필수
-GPU 사용 환경 권장 (4GB VRAM 이상)
-이미지 파일이 명확하고 복잡하지 않아야 한다.

▶시스템 요구사항

-Python 3.8 이상
-GPU: NVIDIA GPU, 최소 4GB VRAM (GTX 1650 이상 권장)
-CUDA 지원
-Ollama 설치 필수

▶설치 절차

-Ollama 설치 (설치 후 재부팅 권장)
Windows 다운로드 링크(Download Ollama on Windows)

-Gemma 3 모델 다운로드 (터미널에서 아래 명령어로 실행)
ollama pull gemma3:4b

-Python 패키지 설치(터미널에서 아래 명령어로 실행)
pip install ollama

▶실행 방법

1. Ollama 서버 실행(터미널에서 아래 명령어로 실행)
ollama serve

2. 프로그램 실행(터미널에서 아래 명령어로 실행)
python main.py

▶사용 방법

1. 프로그램 시작되면 식재료 이미지 경로를 입력한다.

2. VLM이 이미지에서 식재료를 인식하고 목록을 출력한다.

3. 인식된 식재료를 바탕으로 레시피가 자동 생성된다.

4. 생성된 레시피에 대해 질문 할 수 있다.
예) 조리시간은 얼마나 걸리나요?

5. 종료하려면 엔터키만 입력한다.

▶결과 예시

냉장고 AI 셰프에 오신 것을 환영합니다!
분석할 냉장고 이미지 경로를 입력하세요: sample_image/sample_image.png

식재료 인식 중입니다...
인식된 재료: ['양파', '감자', '당근']

레시피 생성 중입니다... 잠시만 기다려주세요.

생성된 레시피:

[요리명]
채소 볶음밥

[필요한 추가 재료]
- 계란
- 간장
- 참기름

[예상 조리 시간]
약 20분 (재료 손질 10분, 조리 10분)

[조리 순서]
1. 양파, 감자, 당근을 잘게 썰어 준비합니다.
2. 프라이팬에 식용유를 두르고 양파를 볶습니다.
3. 감자와 당근을 넣고 함께 볶습니다.
4. 밥을 넣고 재료와 잘 섞이도록 볶아줍니다.
5. 간장으로 간을 하고 마지막에 계란을 넣어 익힙니다.
6. 불을 끄고 참기름을 약간 둘러 마무리합니다.

[조리 팁]
감자는 미리 데쳐두면 더 부드럽고 빨리 익습니다.
참기름은 마지막에 넣어야 향이 살아납니다.

요리에 대해 궁금한 점을 입력해보세요 (엔터만 입력 시 종료)

질문: 감자는 껍질을 벗겨야 하나요?
답변: 네, 감자는 껍질을 벗긴 후에 사용하시는 것이 좋습니다. 껍질에 흙이나 독성이 있을 수 있습니다.

질문: 조리 시간은 얼마인가요?

답변: 총 조리 시간은 약 20분 정도 소요됩니다.

질문:
종료합니다. 즐거운 요리 되세요!

4. 업무 분장

구 분	학 번	이 름	업 무
팀장	20210868	장세준	-모듈 통합 -보고서 작성 및 최종 발표
팀원	20200679	하도윤	-이미지 분석 후 식재료 출력 구현 -README.md 작성 및 git 업로드
팀원	20210866	임원석	-모듈 통합 -터미널 입·출력 구현
팀원	20210881	하승훈	-프로젝트 시연 및 테스트 -보고서 작성
팀원	20210988	설동화	-분석한 식재료로 레시피 출력 구현
팀원	20220280	한창하	-레시피 출력 후 사용자 Q&A 구현