Assignment 1:

Reflexive Web Agent with Tools Use

Student ID: 113524013, Name: 張祐豪

1. **(10%) Describe your Agentic AI application scenario, including target users, use cases, and problems to be solved.**

Provide a comprehensive description of your application with at least 2-3 specific use cases, clearly defined target users, and concrete problems to be solved.

**1. Introduction**

This application is an **AI-powered search assistant** designed to help users find English learning resources quickly and efficiently. Instead of manually searching through different websites, users can **look up vocabulary, find articles or videos, and get grammar explanations** with just one request.

**2. Target Users**

* **English Learners** – Students and self-learners who want quick access to vocabulary, grammar, and reading materials.
* **Teachers** – Educators who need reliable resources for their lessons.
* **Content Creators & Researchers** – Individuals looking for high-quality English content efficiently.

**3. Use Cases**

1. **Looking Up Vocabulary** – A student types a word, and the AI fetches its meaning, example sentence, and a Yahoo Dictionary link instantly.
2. **Finding Relevant Articles & Videos** – A teacher searches for “climate change,” and the AI provides **news articles and educational videos** from trusted sources.
3. **Getting Grammar Explanations** – A learner asks about “passive voice,” and the AI finds a clear explanation from a grammar website.

**4. Benefits**

* **Saves Time** – No need to search multiple websites manually.
* **Simplifies Learning** – Vocabulary, articles, videos, and grammar are available in one place.
* **Useful for Everyone** – Helps both learners and educators.

This application **makes English learning easier and more efficient** by automating the search process and providing **clear, organized results**.

1. **(10%) Analyze at least 2 potential technical challenges in implementation and propose preliminary solutions.**

For each technical challenge, provide detailed analysis including impact assessment and step-by-step solution proposals with feasibility evaluation.

**1. Challenge: Configuring prompts.py and Maintaining Search Functionality**

**Analysis:**

Initially, I attempted to modify the prompts.py file to optimize the search results, but this led to unintended disruptions in the search process. After testing, I decided **not to alter the original prompts** to ensure search stability. Instead, I focused on **improving the task management system** by adding several functions, such as:

* **Loading .env configurations automatically** to streamline API key and file path management.
* **Introducing dynamic task creation (add\_task)** to allow users to input specific queries for word lookup, grammar explanations, and related videos.
* **Implementing clear\_task\_file** to prevent redundant data accumulation without requiring manual file deletions.
* **Enhancing load\_tasks and process\_tasks** to better handle task execution and ensure smooth workflow.

**Impact:**

* Keeping the prompts.py file unchanged **preserves search accuracy**, ensuring queries are properly formatted for different websites.
* The added functions **simplify user interaction**, allowing for **dynamic task input and execution** without modifying core files.
* Removing **unnecessary JSONL deletions** prevents accidental loss of important task data.

**Solution:**

1. **Maintain original prompts.py settings** while optimizing the workflow externally.
2. **Use environment variables (.env)** to manage API keys and file paths, reducing hard-coded dependencies.
3. **Enable dynamic task creation** with verification prompts to prevent incorrect task execution.

**Feasibility:**

* These changes **do not interfere with the original AI agent's search mechanism**, ensuring continued functionality.
* By shifting focus to **task automation and user input management**, the system becomes **more flexible and efficient**.

**2. Challenge: Optimizing Task Execution and Automation**

**Analysis:**

Initially, the program required **manual task input** each time it was executed, and .jsonl files had to be deleted manually to avoid duplicate processing. This approach was inefficient, leading to potential errors.

To **improve automation**, I introduced:

* **Automatic .jsonl clearing at startup** to ensure fresh tasks each session.
* **A structured task creation system (add\_task)** that verifies user input before saving tasks.
* **A command-based interface** that allows users to dynamically add, review, and execute tasks (a for adding, q for executing, x for exit).

**Impact:**

* **Eliminates the need for manual file cleanup**, reducing the risk of executing outdated tasks.
* **Enhances user control over task creation and execution**, minimizing input errors.
* **Ensures a smooth, step-by-step process** where users confirm task details before saving.

**Solution:**

1. **Automate JSONL management** to keep task data clean without unnecessary deletions.
2. **Create a structured user input process** that prompts users for confirmation before storing tasks.
3. **Use a command-based system** to guide users through task creation and execution seamlessly.

**Feasibility:**

* Implementing these solutions **requires minimal additional computation** and **leverages existing Python libraries**.
* The approach **improves system efficiency** without modifying the core AI agent's search behavior.

1. **(20%) Explain how your system implements the complete cycle of environment perception, decision making, and action execution.**

Detail the complete workflow of your system, demonstrating how each component interacts within the perception-brain-action cycle.

My Agentic AI system follows a structured process that mirrors human web browsing behavior. It consists of three main stages: environment perception, decision-making, and action execution. Each stage is crucial for ensuring smooth task completion.

**Step 1: Environment Perception**

The system first gathers information about the webpage using Selenium. It identifies key elements such as:

* Search boxes (for entering queries)
* Buttons (for submitting forms)
* Clickable links (for navigation)
* Text content (for analyzing page structure)

This perception stage ensures that the AI agent understands the webpage layout before interacting with it.

**Implementation in Code:**  
The system dynamically creates tasks based on user input. For example, if a user wants to look up a word, it constructs the corresponding search URL.

def add\_task():

    """動態新增學習任務"""

    while True:

        print("📝 你可以選擇以下學習任務類型：")

        print("1️⃣ 查單字")

        print("2️⃣ 找相關影片")

        print("3️⃣ 問文法問題")

        choice = input("請輸入選項 (1/2/3)：")

        if choice == '1':

            word = input("請輸入你想查的英文單字：")

            task = {

                "web\_name": "英語學習助手",

                "id": "word\_lookup",

                "ques": f"請幫我查 '{word}' 的意思，並給我一個例句。",

                "web": f"https://tw.dictionary.search.yahoo.com/search?p={word}"

            }

        elif choice == '2':

            topic = input("請輸入你想閱讀的主題 (e.g., 環保、科技、旅遊)：")

            task = {

                "web\_name": "英語相關影片",

                "id": "video\_suggestion",

                "ques": f"請推薦一篇與 '{topic}' 相關的英文刊物。",

                "web": f"https://tw.voicetube.com/channels/news-and-current-affairs"

            }

        elif choice == '3':

            grammar = input("請輸入你想學的文法概念 (e.g., 現在完成式、被動語態)：")

            task = {

                "web\_name": "文法小老師",

                "id": "grammar\_help",

                "ques": f"請找搜尋出 '{grammar}'，點選到文法頁面。",

                "web": "https://www.ehanlin.com.tw/app/keyword/%E5%9C%8B%E4%B8%AD/%E8%8B%B1%E8%AA%9E/%E5%88%97%E8%A1%A8.html"

            }

        else:

            print("❌ 無效選項，請重新輸入。")

            continue

        # 顯示預覽

        print("\n🔍 預覽任務內容：")

        print(json.dumps(task, ensure\_ascii=False, indent=2))

**Step 2: Decision Making**

After gathering information from the webpage, the system decides what action to take. GPT-4 processes the webpage elements and selects the most appropriate action:

* If a search box is found → The system inputs a query and submits.
* If a button is found → The system clicks it.
* If the page needs scrolling → The system scrolls down to reveal more content.

**Implementation in Code:**  
To ensure correct decision-making, the system loads and processes the user’s task:

def load\_tasks(file\_path):

    """動態讀取 JSONL 檔案"""

    tasks = []

    try:

        with open(file\_path, 'r', encoding='utf-8') as f:

            for line in f:

                tasks.append(json.loads(line))

    except FileNotFoundError:

        print(f"❌ 檔案 {file\_path} 不存在，請檢查路徑。")

    return tasks

def process\_tasks(tasks):

    """處理任務的邏輯"""

    for task in tasks:

        print(f"🚀 正在處理任務：{task.get('ques', '未提供問題')}")

        # 這裡放原本處理任務的邏輯

        time.sleep(1)  # 模擬處理過程

**Step 3: Action Execution**

After making a decision, the system executes the action using Selenium.

* If the action is "Click", it clicks the button.
* If the action is "Type", it inputs text into a search box.
* If the action is "Scroll", it scrolls the page to find more content.

**Implementation in Code:**  
The AI agent performs the necessary action on the webpage:

def main():

    parser = argparse.ArgumentParser()

        # 優先讀取命令列參數，否則從環境變數中讀取

    parser.add\_argument('--test\_file', type=str, default=os.getenv("TEST\_FILE"))

    parser.add\_argument('--api\_key', type=str, default=os.getenv("OPENAI\_API\_KEY"))

    parser.add\_argument('--api\_model', type=str, default=os.getenv("OPENAI\_API\_MODEL", "gpt-4-vision-preview"))

    ##parser.add\_argument('--test\_file', type=str, default='data/test.json')

    parser.add\_argument('--max\_iter', type=int, default=5)

    ##parser.add\_argument("--api\_key", default="key", type=str, help="YOUR\_OPENAI\_API\_KEY")

    ##parser.add\_argument("--api\_model", default="gpt-4-vision-preview", type=str, help="api model name")

    parser.add\_argument("--output\_dir", type=str, default='results')

    parser.add\_argument("--seed", type=int, default=None)

    parser.add\_argument("--max\_attached\_imgs", type=int, default=1)

    parser.add\_argument("--temperature", type=float, default=1.0)

    parser.add\_argument("--download\_dir", type=str, default="downloads")

    parser.add\_argument("--text\_only", action='store\_true')

    # for web browser

    parser.add\_argument("--headless", action='store\_true', help='The window of selenium')

    parser.add\_argument("--save\_accessibility\_tree", action='store\_true')

    parser.add\_argument("--force\_device\_scale", action='store\_true')

    parser.add\_argument("--window\_width", type=int, default=1024)

    parser.add\_argument("--window\_height", type=int, default=768)  # for headless mode, there is no address bar

    parser.add\_argument("--fix\_box\_color", action='store\_true')

    args = parser.parse\_args()

    # 程式啟動時，自動清空 .jsonl 檔案

    clear\_task\_file()

    # 檢查 API 金鑰是否正確設定

    if not args.api\_key:

        raise ValueError("API key is not set. Please set it in the .env file or pass it via '--api\_key' argument.")

    # OpenAI client

    client = OpenAI(api\_key=args.api\_key)

    # 驗證參數

    print(f"Test File: {args.test\_file}")

    print(f"API Key: {args.api\_key}")

    print(f"API Model: {args.api\_model}")

    options = driver\_config(args)

**Full Cycle Recap**

The entire process follows a structured loop:

1. **Environment Perception:**
   * Uses Selenium to extract web elements and page structure.
   * Identifies buttons, search boxes, and text content.
2. **Decision Making:**
   * GPT-4 analyzes the page and determines the best action.
   * Uses predefined rules to match the task with the best interaction.
3. **Action Execution:**
   * Uses Selenium to simulate clicks, typing, and scrolling.
   * Updates the system state and loops back to perception for the next step.

By following this **Perception → Decision Making → Action Execution** cycle, the AI agent efficiently automates web-based learning tasks while maintaining flexibility.

1. **(30%) Design and execute 3 test tasks, analyze the results, and propose potential improvements based on the current implementation.**

Document the execution of three test cases with comprehensive analysis of results and specific improvement suggestions.

**1. Vocabulary Search**

For vocabulary search, I used a small trick by incorporating {word} directly into the search query.

        if choice == '1':

            word = input("請輸入你想查的英文單字：")

            task = {

                "web\_name": "英語學習助手",

                "id": "word\_lookup",

                "ques": f"請幫我查 '{word}' 的意思，並給我一個例句。",

                "web": f"https://tw.dictionary.search.yahoo.com/search?p={word}"

            }

The search functionality works smoothly without any issues.

**2. Video Search**

Initially, I wanted to retrieve English articles from Taiwan News for reading material. However, I found the website to be extremely **"messy"**—filled with excessive ads and unrelated recommendations, which was frustrating. Since I frequently use VoiceTube for English teaching, I decided to switch to searching for relevant videos instead.

    elif choice == '2':

            topic = input("請輸入你想閱讀的主題 (e.g., 環保、科技、旅遊)：")

            task = {

                "web\_name": "英語相關影片",

                "id": "video\_suggestion",

                "ques": f"請推薦一篇與 '{topic}' 相關的英文刊物。",

                "web": f"https://tw.voicetube.com/channels/news-and-current-affairs"

            }

This adjustment significantly improved the quality of the search results.

**3. Grammar Search**

Grammar is a frequent challenge for my junior high school students. To address this, I integrated a search function with the Hanlin educational website. However, I quickly encountered a major issue—AI often failed to click on the correct grammar rules or took screenshots at the wrong moments, making the process frustrating.

After nearly **20 test runs**, I finally managed to fine-tune the system for better accuracy.

        elif choice == '3':

            grammar = input("請輸入你想學的文法概念 (e.g., 現在完成式、被動語態)：")

            task = {

                "web\_name": "文法小老師",

                "id": "grammar\_help",

                "ques": f"請找搜尋出 '{grammar}'，點選到文法頁面。",

                "web": "https://www.ehanlin.com.tw/app/keyword/%E5%9C%8B%E4%B8%AD/%E8%8B%B1%E8%AA%9E/%E5%88%97%E8%A1%A8.html"

            }

Although refining the prompts was challenging, the final version successfully retrieves grammar rules more accurately.

This project took a lot of my time and resources, and I truly hope my professor will appreciate my report. I'm also very grateful for the opportunity to work on this assignment, as it allowed me to build a meaningful project and share it on GitHub during my time as a master's student at National Central University. The entire process was both challenging and rewarding, and I sincerely thank my professor for providing this valuable learning experience.