Assignment 3:
Agents with Memory Recall via RAG

Student ID: 113524025, Name: 郁宸瑋

1. (20%) Document Processing — Preparing Operation Manuals for Retrieval Augmented Generation Systems

How can operation manuals in various formats (e.g., PDF documents, scanned images, web pages) be processed and structured for use in a RAG system? Please describe how you would:

- Extract and convert text from different formats (e.g., PDF to Markdown, OCR for images)
- Apply chunking strategies to ensure optimal retrieval granularity
- Build a vector database using embeddings suitable for downstream retrieval
- Handle metadata (e.g., section titles, timestamps) to improve retrieval relevance and ranking

① 文字轉換（Text Extraction and Conversion）

在 DocumentConverter 類別中，加入了 OCR 功能以支援無法直接擷取文字的 PDF 頁面（例如掃描圖檔）：

```
if use_ocr and not text.strip():
    img = page.to_image(resolution=300).original
    text = pytesseract.image_to_string(img, lang=ocr_lang)
```

說明：

- 若 PDF 頁面為掃描圖片無法提取文字，則自動使用 pytesseract 執行 OCR
- 支援中日文(chi_tra+jpn），符合 RAG 處理多語手冊需求

② 切段策略（Text Chunking）

在 TextSplitter 類別中，改用 TokenTextSplitter，並設置 chunk_size = 500, chunk_overlap = 100 以控制分段長度與重疊：

```
self.splitter = TokenTextSplitter(
    chunk_size=chunk_size,
    chunk_overlap=chunk_overlap,
    length_function=len,
)
```

說明：

- 將一份文件依 Token 長度切成小段，並保留部分重疊以強化語境銜接
- 選擇 token 為單位更能準確對應語言模型的語意單元

③ 建立向量資料庫（Vector Indexing）

在 RAGEngine 的 index_document()中，依據檔案格式分為 PDF 與 Markdown，皆會先加上時間戳 timestamp 進入 metadata：

```
metadata["timestamp"] = datetime.fromtimestamp(os.path.getmtime(pdf_path)).isoformat()
```

接著依據是否有 TOC(目錄）進行不同分段：

```
if toc:
```

```
            for level, title, start_page, end_page in toc:
                text = ""
                for page_num in range(start_page, end_page + 1):
                    text += doc[page_num].get_text()
                section_metadata = metadata.copy()
                section_metadata.update({
                    "section": title,
                    "level": level,
                    "page_range": f"{start_page + 1}-{end_page + 1}"
                })
                documents.append(Document(page_content=text, metadata=section_metadata))
        else:
            pages = converter.pdf_to_text(pdf_path)
            for page_num, text in pages:
                page_metadata = metadata.copy()
                page_metadata["page"] = page_num
                documents.append(Document(page_content=text, metadata=page_metadata))
```

說明：

- 支援 PDF TOC，可依章節切割，提升檢索精度
- 每段內容都會附加完整 metadata，利於後續語意過濾

④ 中繼資料與搜尋強化(Metadata & Search Reranking)

在 RAGEngine.search()中加入 rerank 策略，若查詢字詞出現在章節名稱或 chunk 中，會額外加權排序：

```
        def _rerank(doc):
            score = doc.metadata.get("score", 0)
            if query.lower() in doc.metadata.get("section", "").lower():
                score += 0.1
            if current_intent and current_intent.lower() in doc.page_content.lower():
                score += 0.05
            return -score
```

說明：

- query 若命中 section title(如：「付款設定」、「退貨流程」)，會額外提權
- 也支援帶入「代理人當前意圖」(如上一輪步驟），進一步篩選符合目的的段落

2. (25%) Prompt Engineering — Designing Effective Prompts for RAG-based Browser Agents

How can prompt engineering be used to enhance a RAG-based system that helps browser agents complete complex tasks?

Please explain:

- How to design system-level prompts to guide the model's overall behavior (e.g., role, constraints, language use)
- How to create task-specific prompts that adapt to different user goals and input contexts
- How to integrate retrieved manual content into prompts effectively (e.g., in-context examples, instruction chaining)
- How to manage prompt length, relevance ranking, and hallucination avoidance when working with long manuals or multi-step procedures

① System-Level Prompts 設計

在 InstructionManualGenerator.__init__() 中加入了 system_role 與格式約束：

```
# 系統角色與指示語（System Prompt）
system_role: str = "You are a professional technical document assistant.",
```

另外，支援選項：

- step_tracker：是否標註每步驟編號
- hint_markers：是否加入動作提示（如：現在請搜尋…）

在 _generate_prompt() 中整合使用：

```
sys_block = f"{self.system_role}\n"
sys_block += "請遵守：\n• 回答必須使用繁體中文\n• 步驟以數字編號，動詞開頭
\n"
if self.step_tracker:
    sys_block += "• 每步前標註 'Step <編號>/<總步驟數>'\n"
if self.hint_markers:
    sys_block += "• 如有需要，在步驟前加入提示標記\n"
```

```
Steps:
1. 開啟瀏覽器並前往 https://www.amazon.co.jp/
2. 在搜尋框輸入「アミノバイタル タブレット」並按放大鏡圖示執行搜尋（引自段落 #[1]）。
3. 在左側篩選面板設定價格區間，拖動價格滑桿並點擊「検索」更新結果（引自段落 #[1]）。
4. 在搜尋結果頁瀏覽五個商品，快速比較它們的名稱、價格與星級評分摘要。
5. 點擊評價星數 ≥ 4 星的第一個商品，進入詳細頁。
6. 點擊「カスタマーレビュー」標籤，切換到評論區。
7. 閱讀前五則高評分評論，確認產品真實用戶回饋。
```

說明：

- 設計清楚的角色角色（e.g., document assistant）與語言風格（繁體中文 + 動詞開頭）可有效**穩定模型行為、減少偏誤輸出與風格不一**
- step_tracker 與提示標記則利於代理人逐步規劃與執行

② Task-Specific Prompts 設計

在 run_HW3.py 中依據每個任務生成專屬 prompt：

```
task_goal = task["ques"]  # 從任務中抽取具體問題
manual_text = manual_gen.generate_instruction_manual()

init_msg = (
    "[Instruction Manual]\n" + manual_text +
    "\n\nNow given a task: {task['ques']} On website {website}, "
    "search for the product '{product}' and extract its product name,
website and price..."
)
```

說明：

- 每次任務都根據 user goal 自動構建 query + instruction，具備「任務導向」prompt
- 這種「動態任務建構」能力，能根據 input context 調整 query，非常符合題目要求

③ 整合手冊進入 Prompt（RAG 檢索段落 + prompt 範例）

在 _generate_prompt() 中，整合了檢索來的段落（retrieved results）：

```
doc_block = "【檢索到的相關段落】\n"
for idx, c in enumerate(relevant_chunks, 1):
    doc_block += (
```

```
                f"[{idx}] Section: {c['section']} (p.{c.get('page','?')})\n"
                f"{c['content']}\n\n"
            )
```

加入了 in-context example：

```
        example_block = (
            "【範例】\nTask Goal: 在 Amazon.jp 搜尋『アミノバイタル』...\nSteps:\n"
            "1. 開啟瀏覽器並前往...\n2. 在搜尋框輸入...\n"
        )
```

說明：

- **手冊內容會精簡後串入 Prompt**，作為模型生成依據
- 再加上 **in-context 示例**，讓模型模仿格式、動作順序，強化指令鏈結能力（instruction chaining）

④ Prompt 長度管理、避免幻覺、排名控制

在 InstructionManualGenerator._trim_chunks() 中設計了一段 token 控管機制：

```
        def _trim_chunks(self, chunks):
            total_len = sum(len(c["content"]) for c in chunks)
            while total_len > self.max_prompt_tokens and chunks:
                removed = chunks.pop()
                total_len -= len(removed["content"])
```

也在 RAGEngine 的 search() 加上 rerank： 同 1-④

說明：

- 可依長度自動裁切段落，避免 context overflow
- 對長 chunk 使用摘要(summarize=True)，避免 hallucination
- 對 query 與意圖命中 chunk 做 rerank，提高 relevance

3. (25%) RAG — System Architecture and Query-driven Retrieval for Manual-Guided Task Completion

How can a Retrieval-Augmented Generation (RAG) system be designed to help browser agents complete unfamiliar tasks by leveraging retrieved operation manuals?
Please discuss:

- The end-to-end RAG architecture, from query understanding to retrieval and final generation
- How the system retrieves relevant document chunks based on the current task or the agent's intermediate intent (i.e., what the agent is currently reasoning about or attempting to do)
- Optional post-processing techniques for retrieved content (e.g., reranking, filtering, summarization) to enhance precision
- How to incorporate the retrieved content into the generation process to derive accurate and actionable step-by-step instructions
- Optional design consideration: In RAG-guided interactive browsing tasks (ref Slide 19 or 22), could the assistant's responses be enhanced by including features such as a Step Tracker (e.g., "Step 4 of 9") or an Instruction Cue (e.g., "Now focus on filtering by date")? Discuss whether such elements may help the agent better execute actions, stay aligned with the task flow, and focus on the next appropriate action. Feel free to propose additional interaction design ideas that could improve clarity and guidance for the agent.

You may include examples of previously unachievable tasks and explain how the agent successfully completed them after referencing retrieved instructions.

① End-to-End 架構流程（從 Query → 檢索 → 生成）

1. **Query 來源：**從 task["ques"] 中動態獲得任務目標
2. **檢索流程：**使用 PDFEnhancementPipeline.search() → 呼叫 RAGEngine.search()
3. **生成流程：**使用 InstructionManualGenerator 根據檢索結果，產生操作手冊

```
results = pipeline.search(query=task["ques"], k=5,
current_intent=prev_intent, summarize=True)
manual_gen = InstructionManualGenerator(..., results=filtered, ...)
manual_text = manual_gen.generate_instruction_manual()
```

說明：

- 模組清楚分層（RAGEngine 處理檢索，ManualGenerator 處理生成）
- 支援任務導向的「查詢 → 多段落檢索 → 精準步驟生成」

② 根據 Intent 檢索 Chunk（Query 與中介意圖）

在 RAGEngine.search() 中，實作了 query-based + intent-aware 的雙重加權排序機制：

```
def _rerank(doc):
    score = doc.metadata.get("score", 0)
    if query.lower() in doc.metadata.get("section", "").lower():
        score += 0.1
    if current_intent and current_intent.lower() in
doc.page_content.lower():
        score += 0.05
    return -score
```

說明：

- 檢索會考慮目前 query 與代理人的中介推理階段（current_intent）
- 確保檢索內容能符合「目前正在執行什麼」的語境，利於任務對齊與精準指引

③ Post-processing（重新排序、摘要、過濾）

具備下列三種後處理功能：

1. **Rerank：**如上所示，query + intent 加權
2. **Summarization：**對長 chunk 進行摘要（截取首尾各 200 字）：

```
if summarize and len(chunk) > 400:
    head = chunk[:200]
    tail = chunk[-200:]
    chunk = f"{head} … {tail}"
```

3. **Filter：**支援 metadata 過濾，如限定某個 section 或來源文件

說明：

- 可過濾掉過長、不相關 chunk，提高資訊密度與搜尋精度
- 避免超出 context 長度與不必要的「幻覺內容」

④ 將檢索段落融入 Step-by-step 指令

在 InstructionManualGenerator._generate_prompt() 中，任務生成步驟：

```
doc_block = "【檢索到的相關段落】\n"
for idx, c in enumerate(relevant_chunks, 1):
    doc_block += f"[{idx}] Section: {c['section']}
(p.{c.get('page','?')})\n{c['content']}\n\n"
```

然後透過 generate_instruction_manual() 呼叫 GPT 模型產出：

```
response = self.openai_client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "system", "content":
self.system_role},
              {"role": "user", "content": prompt}],
    temperature=0.3
)
```

說明：

- 模型僅根據檢索段落產生步驟，避免杜撰
- 將 retrieved content → operation steps 明確連結，達成手冊導引

⑤ UI 對齊設計：Step Tracker / Instruction Cue 等交互設計

在 InstructionManualGenerator 中提供兩種設計參數來提升指令清晰度與任務對齊：

```
step_tracker=True      # 顯示 Step 1/5
hint_markers=True      # 顯示「現在請登入」「現在
請點選」等提示
```

會被整合進 prompt 的格式中：

```
if self.step_tracker:
    sys_block += "• 每步前標註 'Step <編號>/<總
步驟數>'\n"
if self.hint_markers:
    sys_block += "• 如有需要，在步驟前加入提示標
記\n"
```

說明：

- Step Tracker 讓使用者與代理人清楚知道目前在任務第幾步
- Instruction Cue 則讓每個操作更明確指向「下一個該做什麼」

⑥ 現在能考量多個產品，包含折扣前後、星級…