Assignment 3:
Agents with Memory Recall via RAG

Student ID: 113527602, Name: Slametian Dewa Tegar Perkasa 石柏楷

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

To prepare operation manuals in varied formats for integration with a RAG system, several technical steps are essential. First, the input documents need to be converted into a uniform and machine-readable format. PDF documents can be parsed using tools like `pdfplumber` for extracting structured text or `PyMuPDF` (fitz) for layout-aware processing. For scanned image documents, Optical Character Recognition (OCR) techniques such as Tesseract or Google Vision API can be employed to transcribe content into text. Web pages can be processed using headless browsers with automation libraries (e.g., Selenium) to extract visible text and DOM structures.

Once the text is extracted, it is crucial to segment it into retrievable units or "chunks." RecursiveCharacterTextSplitter from LangChain or sentence-aware chunkers can be used, setting parameters like chunk size (e.g., 500-1000 tokens) and overlap (e.g., 100-200 tokens) to preserve contextual coherence. Additionally, structural markers such as section headers, bullet points, and tables of contents should guide chunk boundaries.

The next step involves embedding the chunks into vector space using models such as OpenAI's `text-embedding-3-small` or `bge-m3` from HuggingFace. These embeddings are stored in a vector database such as Chroma, FAISS, or Weaviate. Metadata, including section titles, page numbers, and timestamps, should be attached to each chunk to support metadata-aware retrieval. These attributes can be indexed to improve ranking relevance using filtering or hybrid search strategies during the retrieval phase.

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

Prompt engineering plays a pivotal role in steering the behavior of a RAG-enhanced browser agent. At the system level, prompts should define the agent's persona (e.g., "You are WebVoyager, a robotic web agent"), operational constraints (e.g., single action per iteration), and formatting rules for responses (e.g., Thought/Action format). This ensures consistency in interaction and aligns generation behavior with user expectations and API requirements.

Task-specific prompts should dynamically adapt to the current objective, such as "searching for a PDF on ArXiv" or "filtering restaurant reviews by date." This can be done by embedding the goal within a task description and combining it with contextual cues from the current iteration (e.g., the page's DOM or accessibility tree).

To effectively integrate retrieved manual content, prompt templates can include a dedicated [Manuals and QA Pairs] section. Instruction chaining can be implemented by clearly enumerating each retrieved step and prompting the model to match them with observable elements. Techniques such as in-context learning and few-shot demonstrations can further enhance grounding.

Prompt length must be managed by clipping message histories and truncating manuals using tokenization tools (e.g., `tiktoken`). Ranking retrieved passages by relevance score before insertion and filtering for redundancy or contradiction help reduce hallucination. Optionally, context-aware rerankers or LLM-based summarizers can compress retrieved steps before inclusion.

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.

The architecture of a RAG system for browser agents consists of several integrated components. The pipeline begins with a user query or task goal, which is encoded and passed to a retriever module. This module performs vector similarity search over an indexed corpus of chunked operation manuals. Depending on the implementation, retrieval can be query-only or query + intent-driven (using intermediate states).

The retrieved content is optionally reranked using relevance feedback or heuristics such as document metadata (e.g., filtering by matching section titles). Summarization or compression may be applied using lightweight models (e.g., DistilGPT2 or a summarization head) to reduce token load.

In the generation phase, the system constructs a prompt that combines the user goal, current webpage context, and retrieved manual snippets. This guides the LLM (e.g., GPT-4o) to produce step-by-step instructions aligned with real UI elements.

To enhance agent alignment with the multi-step process, features like a Step Tracker (e.g., "Step 4 of 9") or Instruction Cues (e.g., "Now focus on filtering by date") can be embedded in the prompt or returned as auxiliary outputs. These cues maintain temporal grounding and support better adherence to the manual. Other ideas include visual anchors (if vision is enabled), backtracking strategies when failure is detected, and hierarchical memory layers that track task progress.

These architectural features allow the agent to reason about previously unachievable tasks, such as extracting precise metadata from technical manuals or navigating deeply nested interfaces. After integrating retrieved instructions, such agents demonstrate significantly improved task success and reduced hallucination.