Part 3: Agentic System Design - The News Research Agent

This document outlines the design for an AI agent capable of leveraging our previously built NLP tools to perform complex research tasks. Instead of relying on a static dataset, this agent will use a live web search tool to gather real-time information. The agent's intelligence and planning capabilities will be powered by the Gemini API.

A. Scenario

The agent operates as a **Real-Time News Research Assistant**. Its primary function is to help a user quickly understand and synthesize information from the live web. The user can ask complex, multi-step questions about current events, and the agent's job is to search the web, extract key information from relevant sources, and provide a consolidated, up-to-date answer.

Example Use Cases:

- **Student:** A student researching a topic could ask, "What are the latest developments in AI regulation as of this week?"
- **Analyst:** A financial analyst might query, "Summarize the market's reaction to the most recent Federal Reserve announcement."
- **Curious User:** Anyone could ask, "Find me some positive news stories about new technology that were published today."

B. Agent Architecture

The agent is designed around a **Plan-and-Execute** framework, orchestrated by the Gemini model.

1. Agent's Goal:

> To receive a user's natural language query, search the live web for relevant and timely articles, analyze the content, and generate a comprehensive, synthesized answer that directly addresses the query.

2. Agent's Tools:
The agent will have access to a set of Python functions that it can choose to call. The Gemini model will be instructed on how to use these tools.

Tool Name	Description	Python Implementation
web_searcher	Searches the web for real-time information, news	A function that calls a search engine API (e.g., Google

	articles, or data relevant to a given query. Returns a list of URLs and snippets.	Search API) to get live results.
content_fetcher	Fetches the full text content from a given URL.	A function that uses a web scraping library (e.g., BeautifulSoup) to extract the main article text from a webpage.
information_extractor	Extracts key named entities (like People, Organizations, Locations) from a given piece of text.	The extract_named_entities function built in Part 2 using spaCy.
text_analyzer	Reads one or more articles and generates a concise summary or synthesizes an answer to a specific question based on their content.	A function that calls the Gemini API with a specific prompt to perform summarization or question-answering on the provided text.

3. Reasoning and Planning Strategy (The Agent Loop):

The agent's core logic follows a loop where Gemini acts as the planner and synthesizer.

- **Step 1: Parse Query & Plan**
  - The user's query (e.g., "Summarize recent news about the company 'SpaceX'") is sent to the Gemini API.
  - The prompt to Gemini includes the user's query and the list of available tools.
  - Gemini's task is to create a step-by-step plan. For example: Plan: 1. Use web_searcher to find recent news about 'SpaceX'. 2. Use content_fetcher on the top 3 URLs. 3. Take the fetched text and use text_analyzer to summarize the key findings.
- **Step 2: Execute Plan**
  - The Python code receives and parses the plan from Gemini.
  - It calls the specified tool functions in the correct order, passing the output of one step as the input to the next.
- **Step 3: Synthesize Final Answer**
  - The results from the tool executions (the "evidence") are collected.
  - This evidence is sent back to Gemini in a final prompt, asking it to formulate a comprehensive, user-friendly answer.

- **Step 4: Respond**
    - The agent presents Gemini's final synthesized response to the user. The agent can also be enhanced with memory to handle follow-up questions by retaining the context of the conversation.

Agent Architecture: Decision Logic & Tool Chaining

This document provides a detailed explanation of the decision-making process and tool-chaining logic for the News Research Agent, as depicted in the provided architecture diagram.

Explanation of Decision Logic

The agent's "brain" is the **Gemini API**. It does not follow a hard-coded set of if/else conditions. Instead, its decision-making is dynamic and based on the Large Language Model's ability to reason and plan.

1. **Goal-Oriented Planning:** When the agent receives a user query (e.g., "Summarize the latest news about AI regulations in Europe"), the **Agent Orchestrator** sends this query along with a manifest of available tools (web_searcher, content_fetcher, etc.) to the Gemini API. Gemini's primary task is to understand the user's *intent* and create a logical, step-by-step plan to achieve it.
2. **Tool Selection:** Gemini decides which tool is most appropriate for each step of its plan.
    - If the query requires current information, it will select web_searcher.
    - If the plan involves reading the content of a webpage, it knows it must use content_fetcher with a URL from the previous step.
    - If the user asks to identify key players, it will select information_extractor.
    - For summarization or synthesis, it will choose text_analyzer.
3. **Synthesis and Final Response:** After the tools have been executed, the collected data ("evidence") is passed back to Gemini. At this stage, the model's decision logic shifts from planning to synthesis. Its goal is to formulate a coherent, well-written, and human-readable answer that directly addresses the user's original query, based only on the evidence provided by the tools.

Explanation of Tool Chaining

Tool chaining is the process of linking the output of one tool to the input of another. This is crucial for executing multi-step plans. The **Agent Orchestrator** manages this flow.

Example Scenario:

User Query: "Who are the key people mentioned in recent articles about SpaceX's Starship project?"

1. **Step 1: Planning**
    - **Gemini's Plan:**
        1. web_searcher(topic="SpaceX Starship project news")
        2. content_fetcher(url=https://www.usmle.org/step-exams/step-1)
        3. information_extractor(text=[Text from step 2])
2. **Step 2: Execution & Chaining**
    - The orchestrator executes web_searcher.
        - **Output:** A list of URLs.
    - The orchestrator takes the first URL from the output and uses it as the input for content_fetcher.
        - **Output:** The full text of the article.
    - The orchestrator takes the article text from the previous step and uses it as the input for information_extractor.
        - **Output:** A list of named entities (e.g., [('Elon Musk', 'PERSON'), ('Gwynne Shotwell', 'PERSON')]).
3. **Step 3: Synthesis**
    - The final list of entities is passed to Gemini, which then formulates the final answer: "The key people mentioned in recent articles about the SpaceX Starship project are Elon Musk and Gwynne Shotwell."

High-Level Pseudocode

This pseudocode represents the logic within the **Agent Orchestrator**.

```
function main()
  // Initialize all available tools
  tools = {
    "web_searcher": initialize_web_searcher(),
    "content_fetcher": initialize_content_fetcher(),
    "information_extractor": initialize_entity_extractor(),
    "text_analyzer": initialize_text_analyzer_with_gemini()
  }

  // 1. Get user query
  user_query = get_input_from_user()

  // 2. Generate a plan using Gemini
  // The prompt includes the query and a description of the tools
  plan_prompt = create_planning_prompt(user_query, tools.descriptions)
```

```
    plan_json = call_gemini_api(plan_prompt) // Returns a structured plan

    // 3. Execute the plan
    evidence_context = {} // Stores results from each step
    for step in plan_json.plan:
        tool_to_call = tools[step.tool_name]

        // Resolve arguments using results from previous steps
        resolved_args = resolve_arguments(step.args, evidence_context)

        // Call the tool
        result = tool_to_call.execute(resolved_args)

        // Store the result for subsequent steps
        evidence_context[step.id] = result

    // 4. Synthesize the final answer
    evidence_summary = format_evidence_for_synthesis(evidence_context)
    synthesis_prompt = create_synthesis_prompt(user_query, evidence_summary)
    final_answer = call_gemini_api(synthesis_prompt)

    // 5. Deliver the answer
    display_answer_to_user(final_answer)

end function
```