Project Report

Name: Solome Getachew
Date: June 20, 2025

1. Overview of Methods

This project successfully developed a sophisticated AI agent capable of real-time research and synthesis. The development process was modular, building foundational NLP capabilities first and then integrating them into an intelligent, autonomous system. The key methods employed across the project include:

- **Text Preprocessing:** Standard NLP preprocessing techniques were used to clean and normalize text data for initial analysis. This involved tokenization, lowercasing, stop-word removal, and lemmatization using the NLTK library. These steps were crucial for reducing noise and preparing the text for feature extraction and modeling.
- **Information Extraction:** Two primary methods were implemented for information extraction:
  - **Rule-Based Extraction:** Regular expressions (regex) were used for simple, pattern-based extraction, such as identifying dates in a standardized format.
  - **Named Entity Recognition (NER):** The spaCy library's pre-trained en_core_web_sm model was used to identify and categorize entities such as persons, organizations, and locations. This provided a structured way to pull key information from unstructured text.
- **Document Summarization:** An abstractive summarization model (google-t5/t5-small) from the Hugging Face Transformers library was implemented. This demonstrated the ability to generate fluent, human-like summaries that capture the essence of a document, rather than just extracting key sentences.
- **Agentic AI Framework:** The final agent was architected using a **Plan-and-Execute** framework orchestrated by the **Gemini API**. This approach leverages the advanced reasoning capabilities of a large language model (LLM) to dynamically create and execute multi-step plans based on a user's query and a set of available tools.

2. Discussion of Results, Challenges, and Model Performance

Results & Performance:
The performance of the final agent is qualitatively high. By leveraging the Gemini API as its core reasoning engine, the agent can understand complex user intent, formulate logical plans, and synthesize information from multiple sources into a coherent answer. The tool-based design proved effective, allowing the agent to interact with the live web and provide up-to-date, relevant information. The modularity of the design means that new tools can be

easily added to expand the agent's capabilities in the future.
Challenges:
Several challenges were encountered during the project:

- **Real-World Web Scraping:** The content_fetcher tool faces the primary challenge of modern web development: not all websites are easily scrapable. JavaScript-heavy sites or sites with anti-bot measures can prevent simple content extraction. A more robust implementation would require tools like Selenium or Playwright.
- **Tool Reliability:** The overall success of the agent is highly dependent on the reliability of its tools. An error in the web search or content fetching step can cause the entire plan to fail.
- **Prompt Engineering:** Crafting the right prompts for the Gemini API to generate a valid, executable JSON plan was an iterative process. The model needs clear instructions and examples to perform reliably.

3. Full Explanation of the Agent Use-Case, Workflow, and Architecture

Use-Case:
The agent serves as a Real-Time News Research Assistant. It empowers users to go beyond simple keyword searches and ask complex questions about current events. For example, a user can ask, "Summarize the key arguments from articles published this week about the future of renewable energy," and receive a synthesized answer rather than just a list of links.
Workflow:
The agent operates on a dynamic, four-step workflow for each query it receives:

1. **Plan:** Gemini analyzes the user's query and the available tools (web_searcher, content_fetcher, etc.) to generate a JSON-based, step-by-step plan.
2. **Execute:** A Python-based orchestrator parses the plan and calls the necessary tools in sequence.
3. **Chain:** The output from each step is automatically fed as input to the next, creating a logical chain of actions (e.g., search -> fetch -> analyze).
4. **Synthesize:** All the information gathered during the execution phase ("evidence") is passed back to Gemini, which then crafts a final, comprehensive answer for the user.

Architecture:
The system is built on a decoupled, tool-based architecture. The core logic resides in the Agent Orchestrator, which communicates with the Gemini API for planning and synthesis. The Toolbox is a collection of Python functions that provide the agent's capabilities. This design is highly scalable and maintainable, as the agent's reasoning (the LLM) is separate from its capabilities (the tools).