Web Research Agent

Overview

This project involves creating a research agent that can process user queries by analyzing, searching, scraping, and synthesizing content from the web. The agent leverages multiple external tools, including Google Search API, web scraping, and OpenAI's LLMs, to generate useful research results, handle errors gracefully, and provide high-quality summaries.

Features

- **Query Analysis**: The agent analyses user queries to break them into subcomponents and generate optimized search terms.
- **Web Search**: The agent performs web searches using the SerpAPI to gather relevant information.
- **Web Scraping**: The agent scrapes relevant data from websites that allow it, extracting key content for synthesis.
- **Content Synthesis**: After gathering data, the agent synthesizes the information into a coherent summary, resolving contradictions and offering references.

Step-by-Step Installation Guide

Prerequisites

- Python 3.8+
- Streamlit
- OpenAI API Key
- SerpAPI Key

1. Clone the Repository

```
git clone https://github.com/kiruba11k/web-research-agent.git

cd research-agent
```

2. Set Up a Virtual Environment

```
python -m venv venv

source venv/bin/activate  # On Windows use `venv\Scripts\activate`
```

3. Install Required Dependencies

Make sure you have all the required libraries.

```
pip install -r requirements.txt
```

4. Set Up API Keys

Add your API keys in .env

```
GROQ_API_KEY = "your_openai_api_key"
```

```
SERPAPI_KEY = "your_serpapi_key"
```

5. Run locally

Once the agent is set up, you can run the app using Streamlit:

```
streamlit run app.py
```

Agent Structure

The agent follows a modular structure, with several nodes defined for different tasks:

- **Query Analysis:** This node processes the user's query to identify the intent and generates optimized search terms. It uses OpenAI's LLM for analyzing the query.
- **Web Search:** This node performs web searches using the SerpAPI, leveraging the optimized search terms from the query analysis.
- **Web Scraping:** This node scrapes content from the websites obtained through the web search, adhering to robots.txt to respect scraping rules.
- **Content Synthesis:** This node combines the content scraped from multiple sources and generates a human-readable summary using LLMs.
- **Final Output:** Displays the synthesized summary to the user.

How It Works

Query Analysis (LLM)

- **Input:** User's query
- **Output:**

  Query intent

  Subcomponents

  Optimized search terms
- **How it's used:** Guides the Web Search stage to find more precise results.

Web Search (SerpAPI)

- **Input:** Optimized search terms
- **Output:** List of URLs and search result snippets
- **How it's used:** Selects relevant sources to scrape for content.

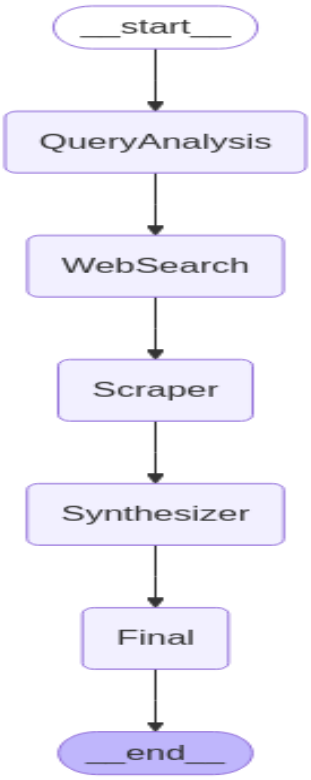
Web Scraping (BeautifulSoup)

- **Input:** URLs from the web search
- **Output:** Raw extracted website content
- **How it's used:** Collects material for the content synthesis phase.

Content Synthesis (OpenAI LLM)

- **Input:** Scraped content from multiple websites
- **Output:** Human-readable synthesized summary
- **How it's used:** Provides final answer to the user, including references.

Flow Diagram



1. Agent Structure

The agent is **built as a multi step pipeline** (like a flowchart) using **LangGraph's StateGraph**, and processes a user's research query across several stages:

Step	Description
1. Query Analysis	Understand the user's query intent, break it down, and generate optimized search terms.
2. Web Search	Use **SerpAPI** (Google Search API) to find relevant webpages based on the search terms.
3. Web Scraping	Scrape the content of the top search results (with respect to robots.txt) using BeautifulSoup and pandas.
4. Content Synthesis	Summarize the scraped content into a final concise answer using LLM (Groq/OpenAI).
State Management	The AgentState TypedDict is used to track important information across steps (messages, plans, results, errors, memory).

---

2. Prompt/Instruction Design

At the **Query Analysis** step, a clear and tightly structured **prompt** is crafted:

- **Role Instruction:** Tells the AI it is a *query analysis expert*.
- **Task Instruction:**
    - Identify the *intent* (factual, opinion, news, etc.).
    - Break down the query into *sub-components*.
    - Generate *optimized search terms*.
- **Strict Output Format:** Forces the AI to return in **strict JSON**, making parsing easy and predictable.

Example prompt sent:

```
You are a query analysis expert. Given this research query: "..."
- Identify the intent (e.g., factual, opinion, news, history)
- Break it into subcomponents
- Generate optimized Google search terms
Return your answer in strict JSON format.
```

This avoids hallucinations and makes later steps more reliable.

---

3. External Tool Connections

The agent connects to external services and libraries smartly:

External Tool	Purpose
Groq OpenAI Client (client)	Interact with LLMs like Llama 3 for prompt-based generation (via API keys).
SerpAPI	Perform real-time Google searches programmatically.
Requests	Fetch webpage HTML for scraping.
BeautifulSoup	Parse and extract clean text content from webpages.
Pandas	Read tables (HTML <table>) from webpages automatically.
LangGraph	Manage flow between different agent states (processing steps).
Streamlit	Frontend display (buttons, toggle views, etc.) for users.

API keys (GROQ_API_KEY, SERPAPI_KEY) are **securely stored** in st.secrets, following **best practices**.

---

4. Error Handling and Unexpected Situations

The code includes **robust error handling** at every critical step:

Step	Error Handling Strategy
Query Analysis	If LLM response is empty, invalid, or non-JSON, catches error and logs raw output for debugging.
Web Search	Checks if search returns no results; adjusts search based on query intent (like adding "review" or "Reddit" for opinions).
Web Scraping	Respects robots.txt; uses multiple user-agents; retries on network errors; skips broken sites; handles scraping failures gracefully.
Content Synthesis	If user query or scraped content is missing, a fallback error message is raised.
Global Error Handling	In every node, errors are caught and stored in the state["error"] field, without crashing the whole system.

- Scraping uses **random user-agents** to avoid detection.
- **Respect for robots.txt** is built-in to avoid illegal scraping.
- **Retries** are configured for failed HTTP requests.
- **Timeouts** ensure the agent doesn't hang on slow websites.

Design Details

- **Memory Management:**
    - Stores important intermediate data like query plans inside a memory dictionary in the state object.
    - This allows later steps (search, scraping, summarization) to **access earlier results**.
- **Content Filtering:**
    - Only saves meaningful content (minimum 150 characters) to ensure garbage data is avoided.
    - Text is trimmed to 2000 characters per page to optimize token usage for LLM summarization.
- **Frontend Touch:**
    - Toggle buttons and result containers are styled using Streamlit's st.markdown with custom HTML/CSS.
    - Adds user-friendly toggles to hide/show raw search results if needed.

Error Handling

Error handling is integrated at every step to ensure the agent functions robustly. For example, if a query analysis fails, the error is captured and the user is notified:

```
except json.JSONDecodeError as e:
    state["error"] = f"JSON parsing error: {str(e)}\nRaw output:\n{raw_output}"
```

This helps in providing graceful feedback instead of crashing the system.

Test Cases

To prove that your agent works under different conditions and handles errors correctly.

- Normal successful query
- Empty LLM response (query analysis fails)
- No search results returned
- Scraping failure (bad website/blocked)

Test Cases Samples

```
[
  {
    "name": "Normal Successful Query",
    "input_query": "What are the latest trends in Artificial Intelligence for healthcare?",
    "expected_behavior": "Agent should perform query analysis, search, scrape and synthesize a detailed answer.",
    "simulate_error": false
  },
  {
    "name": "Empty LLM Response in Query Analysis",
    "input_query": "How does AI impact education?",
    "expected_behavior": "Agent should catch an error after query analysis step when LLM returns nothing.",
    "simulate_error": "empty_llm_response"
  },
  {
    "name": "No Search Results",
```

```
  "input_query": "Explain unicorn startup dynamics on Mars colony.",
  "expected_behavior": "Agent should catch an error when search returns no results.",
  "simulate_error": "no_search_results"
 },
 {
  "name": "Scraping Failure",
  "input_query": "Recent breakthroughs in nuclear fusion energy.",
  "expected_behavior": "Agent should handle scraping errors (bad URLs or blocked pages) gracefully without crashing.",
  "simulate_error": "scraping_failure"
 }
]
```

Conclusion

This research agent is designed to help users gather and synthesize information from the web based on a query. It is structured for flexibility, scalability, and effective error handling, making it a valuable tool for research tasks.

Links

Github : https://github.com/kiruba11k/web-research-agent.git

Live Demo : https://mywebresearchagent.streamlit.app/

Loom Video : https://www.loom.com/share/d976676fca774adcb4d79077fcc3e09f?sid=6a6606e8-5b8a-4ccc-a7a3-69607d046e7e