Web Research Agent Documentation

1. Agent Structure

The agent is built as a Streamlit web application orchestrating several backend processes. Its structure can be broken down into the following key components:

1. **User Interface (Streamlit):**
    - Provides the web front-end using the streamlit library.
    - Handles user input for the research query (st.text_area).
    - Displays status messages, progress indicators (st.spinner, st.info, st.success, st.warning, st.error), and the final generated report (st.markdown).
    - Includes sidebar elements for information (st.sidebar).
    - Triggers the research process via a button (st.button).

2. **Configuration and Setup:**
    - Loads API keys securely from a .env file using python-dotenv.
    - Defines global constants like MAX_CHARS_PER_PAGE and MAX_RESULTS_TO_PROCESS.
    - Configures the Google Gemini client (genai.configure, genai.GenerativeModel) when the user initiates the research.

3. **Core Orchestration (run_research_agent function):**
    - This is the main function that manages the entire research workflow from query input to report generation.
    - It executes the steps sequentially:
        - **Initial Query Analysis:** Performs a basic keyword check to determine if the query seems "News-focused".
        - **Tool Selection:** Chooses between gnews_search (for news) or tavily_web_search (for general search) based on the analysis.
        - **Web/News Search:** Calls the selected search function to retrieve a list of relevant URLs and snippets.
        - **Iterative Content Processing:** Loops through the search results (up to MAX_RESULTS_TO_PROCESS):

        - Performs basic duplicate URL checks.
        - Calls scrape_website_content to fetch and extract text from each URL.
        - Filters results based on scraping success and collects the extracted text, URL, and title.
    - **Context Preparation:** Formats the successfully collected data (URL, title, text) into a structured text block (context) suitable for the LLM.
    - **LLM Synthesis:** Sends the prepared context and the original user query to the Gemini model via a carefully crafted prompt.
    - **Report Generation:** Receives the synthesized report text from Gemini.
    - **Return Result:** Returns the final report string to the Streamlit UI for display.

4. **Tool Functions (Modular Components):**
    - **tavily_web_search(query, api_key, max_results):** Interacts with the Tavily Search API via its Python client to perform general web searches. Returns a list of dictionaries containing search result details (URL, title, content snippet).
    - **gnews_search(query, api_key, max_results):** Interacts with the NewsAPI.org service (using the newsapi-python client library) to fetch recent news articles matching the query. Returns data in the same format as tavily_web_search.
    - **scrape_website_content(url):** Takes a URL, uses requests to fetch the HTML, and BeautifulSoup (with lxml) to parse it. It attempts to extract the main textual content while removing boilerplate (scripts, styles, common navigation tags) and limits the text length. Returns a dictionary indicating success/failure status and the extracted text.

2. Prompt Design (LLM Synthesis)

The effectiveness of the agent heavily relies on instructing the Gemini LLM correctly during the synthesis phase. The prompt engineering strategy focuses on clarity, grounding, attribution, and controlled output.

The prompt constructed within run_research_agent before the final LLM call includes these key elements:

1. **Role Definition (Implicit):** The prompt asks the LLM to "synthesize a comprehensive research report," clearly defining its task.
2. **Contextual Grounding:**
    - The prompt explicitly states: "Based *only* on the following text extracted from various web sources..."
    - All the scraped text is provided within a clearly marked block (Extracted Information: ... ---).
    - Instruction #3 reinforces this: "Use information from the sources provided. **Do not add external knowledge or information not present in the text extracts.**"
    - *Reasoning:* This is critical to prevent the LLM from hallucinating or using its general knowledge instead of the actual research findings. It grounds the response in the gathered evidence.

3. **Task Specification:**
    - Instruction #1: "Carefully analyze the user query ({user_query}) and the provided text..."
    - Instruction #2: "Construct a coherent report that directly addresses the query."
    - *Reasoning:* Ensures the LLM focuses on answering the *specific* user question using the provided context.

4. **Attribution Requirement:**
    - Instruction #4: "When presenting information derived from a specific source, **cite the source URL** (e.g., "According to [URL], ...")."
    - *Reasoning:* Makes the report traceable and allows the user to verify information by visiting the original sources.

5. **Handling Discrepancies:**
    - Instruction #5: "Combine insights logically. If sources conflict, state the conflicting information and attribute it to the respective sources..."
    - *Reasoning:* Guides the LLM on how to handle potentially contradictory information found across different sources, promoting transparency.

6. **Acknowledging Limitations:**
    - Instruction #6: "If the provided text is insufficient to fully answer the query, clearly state the limitations."
    - *Reasoning:* Encourages honesty if the research didn't yield enough information, managing user expectations.

7. **Formatting Instructions:**
    - Instruction #7: "Format the report clearly using markdown (headings, lists, bold text)..."
    - *Reasoning:* Ensures the output is readable and well-structured within the Streamlit interface.

3. External Tool Integration

The agent connects to and utilizes several external tools/libraries:

1. **Tavily Search API (tavily-python):**
    - **Connection:** Initialized via TavilyClient(api_key=tavily_key). Requires the Tavily API key.
    - **Usage:** Called by tavily_web_search function. Takes the search query string as input.
    - **Output:** Returns a list of search result objects. The agent extracts the url, title, and content (snippet) from each result.
    - **Purpose:** Provides general web search results for non-news queries.

2. **NewsAPI.org (newsapi-python):**
    - **Connection:** Initialized via NewsApiClient(api_key=gnews_key). Requires the NewsAPI.org key (loaded into GNEWS_API_KEY variable).
    - **Usage:** Called by gnews_search function. Takes the search query string as input. Uses the get_everything endpoint, sorted by relevancy.
    - **Output:** Returns a dictionary containing articles. The agent extracts the url, title, and description (used as content snippet) from the articles list.
    - **Purpose:** Provides recent, relevant news articles for news-focused queries.

3. **Web Scraping (requests & BeautifulSoup4):**
    - **Connection:** Uses the standard requests.get(url, ...) function to fetch HTML content. Uses BeautifulSoup(response.content, 'lxml') to parse the HTML. Requires lxml to be installed.
    - **Usage:** Called by scrape_website_content function for each relevant URL from search results.
    - **Output:** Returns a dictionary containing the scraping status (Success or Failure), the extracted text (or None), and an optional error message.
    - **Purpose:** Extracts the main textual content from web pages for LLM analysis.

4. **Google Gemini API (google-generativeai):**
    - **Connection:** Configured via genai.configure(api_key=...) and initialized via genai.GenerativeModel(...). Requires the Google API key.
    - **Usage:** Called within run_research_agent for the final synthesis step. Takes the structured prompt (containing the user query and scraped context) as input.
    - **Output:** Returns a response object. The agent primarily uses response.text to get the generated report. Includes fallback logic to check response.candidates if direct text access fails.
    - **Purpose:** Analyzes the collected information in the context of the user query and generates the final, synthesized report.

4. Error Handling and Unexpected Situations

The agent incorporates several mechanisms to handle potential errors:

1. **API Key Issues:**
    - The search functions (tavily_web_search, gnews_search) check if the API key is provided before attempting to initialize the client. If missing, an error is shown via st.error, and an empty list is returned, halting that search path.
    - The Gemini configuration happens within a try...except block in the main Streamlit execution flow, catching potential key-related or configuration errors.

2. **Network Errors:**
    - The scrape_website_content function wraps the requests.get call in a try...except requests.exceptions.RequestException as e: block to catch connection errors, timeouts, etc. An error message is logged via st.error, and a "Failure" status is returned.
    - API client libraries (Tavily, NewsAPI, Gemini) typically have their own internal handling for basic network issues, but severe or persistent issues might still raise exceptions. These are caught by the general try...except Exception as e: blocks surrounding the API calls in the respective tool functions.

3. **API Service Errors:**
    - Uses response.raise_for_status() within scrape_website_content to catch HTTP errors (like 4xx client errors or 5xx server errors) returned by the website being scraped.
    - The try...except blocks around Tavily, NewsAPI, and Gemini calls catch exceptions raised by the client libraries, which could indicate rate limits, invalid queries, authentication failures, or internal server errors from the API provider. Error messages are displayed via st.error.

4. **Scraping Failures:**
    - **Non-HTML Content:** scrape_website_content checks the Content-Type header and returns a "Failure" status if it's not HTML, preventing parsing errors.
    - **Parsing Issues:** Although lxml is robust, malformed HTML could still cause issues. The broad except Exception as e: within scrape_website_content acts as a fallback catch-all.
    - **Content Not Found:** If the heuristics fail to find a <main>, <article>, or similar tag, and the <body> is also unavailable or empty, the function returns a "Failure: Could not find main content area." status.
    - The main loop in run_research_agent checks the status returned by the scraper and only processes data if status == "Success". Failed scrapes are reported via st.warning.

5. **LLM Failures:**
    - The Gemini API call is wrapped in a try...except block.

- If the primary way of accessing the response (response.text) fails or an exception occurs, it includes a nested try...except to attempt accessing the content via response.candidates[0].content.parts[0].text as a fallback, logging a warning if successful.
- If both methods fail, a generic error message is returned as the final_report.

6. **No Results/Content:**
    - If the initial search (Tavily/GNews) returns no results, the agent displays an error via st.error and returns an informative message immediately.
    - If search results are found but scraping fails for all processed URLs (or they yield no text), an error is shown, and a message indicating failure to extract content is returned.

.