**Expert Research Agent Documentation**

**1. Project Overview and Architecture**

The **Expert Research Agent** is an autonomous AI application designed to answer complex user questions by performing real-time web searches. It utilizes the **LangChain framework** to orchestrate a **Large Language Model (LLM)** with a specialized web-search tool, following the **ReAct (Reasoning and Acting)** paradigm for reliable and verifiable results. The agent is exposed to the user via an interactive **Streamlit** web application.

**Key Capabilities**

* **Real-Time Information Retrieval:** Uses the Tavily search engine for up-to-date information.
* **Structured Reasoning:** Employs the ReAct pattern to reason, plan, execute tools, and observe results before generating a final answer.
* **Contextual Output:** Delivers detailed, well-structured answers in **Markdown** format, complete with sources.
* **Secure Deployment:** Utilizes a .env file and .gitignore to protect sensitive API keys.

**2. Technical Stack and Core Libraries**

The agent is built on a modern Python stack. Understanding these libraries is crucial for discussing the project in a technical interview.

| Library | Definition & Purpose | Interview Explanation |
| --- | --- | --- |
| **LangChain** | A framework for developing applications powered by language models. It simplifies the orchestration, chaining, and creation of complex LLM-driven pipelines and agents. | "LangChain acts as the **operating system** for my agent. It allowed me to seamlessly connect the LLM (Gemini), the search tool (Tavily), and the ReAct logic into a unified, executable pipeline." |
| **langchain\_google\_genai** | The LangChain integration library for Google's Gemini models. This is used to initialize the core reasoning engine. | "I used the ChatGoogleGenerativeAI class with **gemini-2.5-flash** as the main brain of the agent. I selected this model for its balance of high capability, speed, and cost-effectiveness for a research-oriented task." |
| **langchain\_tavily** | The LangChain wrapper for the **Tavily Search API**. Tavily is an AI-focused search API optimized for grounding LLM results. | "This library provides the agent's **eyes to the outside world**. It allows the agent to break its knowledge cutoff and access real-time information, which is non-negotiable for a research agent. I configured it to return a maximum of **3 results (max\_results=3)** to keep the context relevant and manageable." |
| **create\_react\_agent** | A core LangChain function that combines an LLM, a list of tools, and a prompt to create an agent that follows the ReAct pattern. | "This function is the **Agent Factory**. It ensures the LLM's output conforms to the Thought-Action-Observation cycle, making the agent's decisions transparent and verifiable, which is a major advantage over a non-agentic LLM call." |
| **AgentExecutor** | The runtime environment that takes the agent and tools and executes the full chain of steps (tool calls, parsing, observation loops) until a final answer is generated. | "The AgentExecutor is the **engine that runs the agent loop**. I configured it with verbose=True for debugging, and crucial parameters like max\_iterations=30 and early\_stopping\_method='generate' to control its behavior and prevent infinite loops." |
| **Streamlit** | An open-source Python framework that allows developers to quickly build interactive web applications for data science and machine learning projects. | "I used Streamlit to build the **user-facing front-end** (app.py). It provides a simple, modern chat interface for users to interact with the agent without needing to touch the code, significantly improving the user experience." |
| **python-dotenv** | A library to load environment variables from a .env file into os.environ. | "This is essential for **security**. It allows me to separate sensitive credentials (like API keys) from the source code, which is critical for maintaining best practices in a production or open-source environment." |

**3. Code Breakdown: research\_agent.py**

This file sets up the complete, runnable research agent.

**A. Initialization and Security**

| Code Section | Purpose | Key Takeaway |
| --- | --- | --- |
| load\_dotenv() | Loads API keys from the .env file. | **Security/Best Practice:** Ensures sensitive keys are never hardcoded. |
| os.getenv(...) Check | Ensures both the GOOGLE\_API\_KEY and TAVILY\_API\_KEY are present before proceeding. | **Robustness:** Prevents runtime errors caused by missing environment variables. |
| ChatGoogleGenerativeAI(...) | Initializes the LLM with model="gemini-2.5-flash" and a low temperature=0.2. | **Model Choice/Strategy:** A low temperature minimizes creative/hallucinatory output, making the agent more focused and factual for a research task. |

**B. Tool and Agent Setup**

1. **Tool Definition:**

Python

search\_tool = TavilySearch(max\_results=3)

tools = [search\_tool]

The agent is intentionally restricted to **one powerful tool** (tavily\_search\_results\_json), simplifying its decision-making and focusing its capability entirely on web research.

1. The ReAct Prompt (The Agent's "Brain"):

The prompt is the most critical part, serving as the agent's operating manual.

* + **System Persona:** "You are an expert research assistant."
  + **Tool Enforcement:** "**You MUST use the 'tavily\_search\_results\_json' tool** for any question requiring up-to-date or external information."
  + **Output Format:** "**Provide a detailed, well-structured answer in MARKDOWN format with sources included.**"
  + **Stopping Criterion:** A crucial instruction was added: "**Once you have sufficient information... you MUST immediately stop... and proceed to the Final Answer.**" This prevents unnecessary and costly search loops.
  + **ReAct Format Instructions:** Explicitly defines the Thought:, Action:, Action Input:, and Observation: structure the LLM must follow, ensuring the agent's logic remains sound and auditable.

1. **Agent Creation and Execution:**

Python

agent = create\_react\_agent(llm, tools, prompt)

agent\_executor = AgentExecutor(agent=agent, tools=tools, verbose=True, handle\_parsing\_errors=True)

The create\_react\_agent function binds the components, and the AgentExecutor manages the entire conversational/tool-calling life cycle.

**4. Code Breakdown: app.py (Streamlit Frontend)**

This file provides the user interface for the research agent.

**A. Setup and Dependency Injection**

* **Import Strategy:** The application imports the pre-initialized agent\_executor from research\_agent\_main.py. This is key: **the agent is initialized only once** at the app's startup, preventing expensive setup on every user interaction.
* **run\_agent\_streamlit Function:** This wrapper function is essential. It takes the user query, calls agent\_executor.invoke({"input": query}), and cleanly returns only the final\_answer from the output key.

**B. User Interface and State Management**

1. **API Key Health Check:** The sidebar uses os.getenv to check for API keys and provides a clear **"API Keys Missing!"** or **"Agent Dependencies Loaded."** status to the user.
2. **Chat History (st.session\_state):** Streamlit's st.session\_state is used to store and manage the conversation history (st.session\_state.messages). This ensures that the chat persists across interactions within the session.
3. **Interactive Loop:**
   * When a user submits a prompt via st.chat\_input, the prompt is added to the history.
   * A **loading spinner** (st.spinner) is displayed while the run\_agent\_streamlit(prompt) function is called.
   * The **final report is displayed** using st.markdown(final\_report), which correctly renders the Markdown-formatted research answer from the agent.
   * The final report is then added to the session state, closing the loop.

**5. Setup and Running Instructions**

**Prerequisites**

1. **Python:** Requires Python 3.9+
2. **API Keys:**
   * **Google API Key:** For the Gemini LLM.
   * **Tavily API Key:** For the web search tool.

**A. Installation**

1. **Create and activate a virtual environment** (recommended).
2. **Install dependencies** using the provided requirements.txt:

Bash

pip install -r requirements.txt

**B. Configuration**

1. **Create a .env file** in the project root directory:
2. # .env file
3. GOOGLE\_API\_KEY="YOUR\_GEMINI\_API\_KEY\_HERE"
4. TAVILY\_API\_KEY="YOUR\_TAVILY\_API\_KEY\_HERE"
5. **Ensure .gitignore is configured** to ignore the .env file for security.

**C. Execution**

1. **Run the Streamlit application:**

Bash

streamlit run app.py

1. Open the local URL displayed in the terminal (e.g., http://localhost:8501) in your web browser.