**The Scenario**

You are tasked with building an advanced autonomous research agent. Given a high-level topic (e.g., a company or a technology), the agent must:

1. **Plan:** Dynamically create a research plan.
2. **Execute:** Use a set of tools in parallel to execute that plan.
3. **Synthesize:** Aggregate the findings into a coherent report.
4. **Self-Correct:** Identify low-confidence claims in its own report, pause to ask a human for guidance, and then perform targeted corrective actions based on the feedback.

**The Core Challenge**

This test evaluates your ability to build a multi-stage, resilient, and dynamic agent. The key difficulties are:

1. **Dynamic Task Generation:** The graph's structure is not entirely predefined. The first step for the agent is to create a "plan" of research tasks, which will then be executed as parallel branches.
2. **Robust Parallel Tool Use:** The agent must execute multiple, different research tasks concurrently (e.g., fetching financial data and searching for recent news). Your solution must gracefully handle potential failures in any of these parallel tasks without crashing the entire process.
3. **Sophisticated Human-in-the-Loop for Correction:** This is the most challenging part. The agent must interrupt its flow to ask for help on *specific, low-confidence points*. The human's corrective feedback must trigger a precise, limited re-execution of a part of the graph (e.g., re-running a web search with a new query) rather than starting the whole process over.

### **Technical & System Requirements**

Your solution must be a self-contained Colab/Jupyter notebook. You are responsible for designing the state objects, agentic nodes, and the conditional logic that connects them.

**1. State Management (**You must design a comprehensive state object (e.g., a TypedDict). This state is the "single source of truth" and should be capable of tracking:

* The initial research topic.
* The generated research plan (e.g., a list of sub-tasks).
* Results from parallel branches (storing both successful data and error messages).
* A list of tasks remaining to be re-executed after human feedback.
* The synthesized draft report.
* Specific, low-confidence claims identified for human review.
* The final, approved report.

**2. Agent Capabilities (to be implemented as graph nodes):**

* **Node:**
  + **Input:** The initial research topic.
  + **Action:** Uses a (mocked) LLM to generate a structured research plan. The plan should be a list of tasks, where each task specifies which tool to use.
  + **Example Output:** plan = ["search\_news: 'NVIDIA new GPUs'", "get\_financials: 'NVDA'"]
* **Logic (Conditional Edge):**
  + **Action:** This is not a node but the primary routing logic after the planner. It takes the plan and invokes the appropriate tool-using researchers in parallel. You will likely use LangGraph's ability to call a list of callables for this.
* **Tool-Using Researchers (Nodes):**You must implement at least two distinct researchers that run in parallel branches. Use mocked functions to simulate tool calls.
  + **:** Simulates using a web search tool (like Tavily).
    - **Mock Logic:** Return a realistic-looking text snippet. For the "Self-Correction" scenario, have it return a low-quality source initially (e.g., "From forum-post.com: ...").
  + **:** Simulates fetching financial data.
    - **Mock Logic:** Return structured data for known public companies (e.g., {'price': 900, 'P/E': 75}). It **must** be designed to raise an exception for specific inputs (e.g., private or fictional companies) to test resilience.
* **Node:**
  + **Input:** The aggregated results from all parallel research branches.
  + **Action:** Must handle cases where some branches return data and others return errors. It should use a (mocked) LLM to:
    - Synthesize a draft report from the available data.
    - Critically, identify and list any claims it deems "low confidence." A claim could be an object like {'claim\_text': '...', 'source': '...'}.
  + **Output:** An updated state with the draft report and a list of low-confidence claims.
* **(Conditional Edge):**
  + **Action:** This is the critical decision point after synthesis.
    - If there are no low-confidence claims, it routes to a final finish node.
    - If there are low-confidence claims, it **interrupts the graph** for human input. LangGraph's checkpointer is essential here.
    - Upon resuming with human feedback, it must intelligently determine the next step. It should not re-run the whole graph. It must identify which task needs to be re-run (e.g., the news\_researcher with a new query) and route back *only* to that specific part of the graph.

### **Required Demonstrations (Measurable Scenarios)**

Your final notebook must clearly and successfully execute the following three scenarios. For each, print the final state to demonstrate the outcome.

**Scenario 1: The "Golden Path" - Full Success**

* **Input:** A public company like "NVIDIA".
* **Expected Behavior:**
  1. The planner creates a plan (e.g., ["get\_financials: 'NVDA'", "search\_news: 'NVIDIA new GPUs'"]).
  2. The Financial and News researchers execute in parallel and both succeed.
  3. The synthesizer creates a report and identifies **no** low-confidence claims.
  4. The workflow completes without human intervention, producing a final report.

**Scenario 2: The "Resilient Path" - Graceful Tool Failure**

* **Input:** A private/fictional company like "The Acme Corporation".
* **Expected Behavior:**
  1. The planner creates a research plan.
  2. The financial\_researcher's tool fails with an exception, while the news\_researcher succeeds.
  3. The graph does not crash. The synthesizer receives the news data and an error object from the financial branch.
  4. It produces a partial report, explicitly stating that financial data could not be retrieved.
  5. The workflow completes, demonstrating resilience.

**Scenario 3: The "Self-Correction Path" - Advanced Human-in-the-Loop**

* **Input:** A complex or ambiguous topic like "The future of cold fusion".
* **Expected Behavior:**
  1. The agent researches the topic. The synthesizer produces a draft but flags a specific claim as low-confidence (e.g., Claim: "A commercial breakthrough is expected by 2030", Source: "unverified\_blog.com").
  2. The graph **interrupts**, presenting the draft and the specific claim for verification.
  3. Your code then simulates a human correction: {"correction\_for\_claim": "...", "new\_instruction": "Re-run news search for peer-reviewed papers on commercial viability"}.
  4. The graph resumes. Your logic must intelligently route this new instruction back to the  **only**. The financial\_researcher (or other tools) must not be re-run.
  5. The news\_researcher re-runs with the new constraint, returns updated info, and the synthesizer creates a final, corrected report that is then approved.
  6. The final output state should reflect this correction loop.

### **Bonus Challenge: Multi-Topic Parallel Orchestration (+20 points)**

Extend your solution to handle a list of topics concurrently.

* **Input:** ["NVIDIA", "The Acme Corporation", "The future of cold fusion"]
* **Task:**
  1. Modify your graph to spawn a "research sub-workflow" for each topic. Demonstrate the ability to run these workflows in parallel.
  2. The system must handle multiple, simultaneous interruptions if needed (e.g., one topic needs human review while another is still processing).
  3. Crucially, your state management must ensure that feedback for one topic is only used to resume the subgraph for that specific topic.
  4. The orchestrator should wait until all topics have been fully researched (including any correction loops) and then aggregate all the final reports into a single collection

**Note:**

If you’re not familiar with jupyter notebooks you can build a console app too.