ChatGPT

Slim Map-Reduce Multi-Agent Workflow for Deep Research Tasks

Overview: This architecture adopts a **“context funnel”** approach – akin to a Map-Reduce strategy – to handle complex research tasks by breaking them into smaller subtasks and aggregating results. Instead of a single monolithic agent struggling with long context and complex planning, a team of specialized agents works in stages. A central **planner** orchestrates multiple **worker** agents in parallel (the “map” phase) and then **refines** and **reduces** their outputs (the “reduce” phase) into a coherent final report [1] [2]. Each agent has a **single responsibility**, making the system easier to manage and more scalable. This design addresses the shortcomings of earlier single-agent or sequential approaches by introducing strategic planning, parallel execution, intermediate summarization, and a feedback loop for quality control [3] [4].

Challenges with a Naive Single-Agent Approach

Previous designs that relied on one agent (or a simple chain) for end-to-end research were **complex and inefficient**. Key issues included:

1. **Context Overload:** Having one agent accumulate all information can exceed token limits and overwhelm its context window. When multiple subtopics are explored, the combined context grows too large to handle directly. This necessitates breaking down and summarizing intermediate results [5]. Without intermediate compression, the agent either truncates important information or fails to integrate everything coherently.

2. **Lack of Role Specialization:** A single agent handling planning, searching, analyzing, and writing blurs responsibilities. It is difficult for one prompt (or one model) to be optimized for all tasks. For example, the style of prompt for planning a research strategy differs from that for summarizing content or critically reviewing a draft. This all-in-one approach violates the Single Responsibility Principle, making the prompt design and debugging very complex.

3. **Sequential Bottlenecks:** A monolithic workflow often proceeds step-by-step through subtopics, unable to parallelize work. This is slow and prevents broad exploration. It also means intermediate results aren’t utilized until the very end, and there’s no structured way to revisit or revise earlier steps. The agent might miss connections between subtopics or fail to revise the plan on the fly. In contrast, a planned parallel approach can explore multiple avenues concurrently [4].

4. **No Systematic Quality Feedback:** Without a separate evaluation step, the final output’s quality depends entirely on the single pass of the agent. If the result is lacking (e.g. missing info or containing errors), there’s no built-in mechanism to iteratively improve it. Relying on the same agent to self-critique in one go is unreliable. A dedicated critique and revise loop is needed to ensure a high-quality report.

These challenges motivated a more modular **map-reduce style architecture** where each phase tackles a subset of the problem, and their outputs funnel into subsequent phases for consolidation and refinement.

Map-Reduce Style Deep Research Architecture

The proposed workflow divides the research task into multiple phases with distinct agent roles. The process flows like a pipeline from planning to execution, refinement, integration, and finally evaluation. Importantly, it uses a **master–worker model** at two key stages (research and refining) to parallelize work and then consolidate it, similar to how Map-Reduce jobs split and aggregate data [2]. Throughout, a **shared state** (common memory) persists information so that intermediate findings can be stored and referenced by later stages [6]. Below we describe each role in the system and its responsibility:

Planner (Master Orchestrator)

The **Planner** is the mastermind that formulates a high-level research plan before any detailed work begins. Rather than diving straight into answering the query, the planner first **strategizes**. It may break the user's query into sub-questions or topics and outline an approach to tackle each. In essence, the planner creates a blueprint or roadmap for the project, often structured as a list of tasks or even a directed acyclic graph (DAG) of interconnected questions [4]. This plan ensures the agent system moves from **tactics to strategy**, meaning it has foresight about the multiple stages needed rather than just reacting step by step [3].

- **Task Decomposition:** The planner may decompose a broad query ("Research AI") into major components. For example, it might identify key angles like *historical evolution of AI*, *current technologies*, and *future trends*. Each becomes a subtask to investigate.
- **Output (Plan):** The planner outputs a structured plan (e.g. a JSON list of tasks, or a DAG structure) that lists these subtopics or questions to be answered. This output serves as the input for the next phase. By doing this upfront, the system treats the problem holistically and can pursue multiple avenues in parallel, rather than linearly solving one aspect at a time [7] [1].

Notably, the planner's job is purely high-level planning. It does *not* fetch information itself; it delegates that to workers. This specialization keeps the planning prompt focused (e.g. "Given the query, outline a research plan with subtopics X, Y, Z…"). The planner may use a relatively powerful model to ensure a good strategy, but it runs only once at the start (and possibly again if revised).

Router (Dynamic Task Router)

The **Router** agent is an optional but useful role that routes or adjusts tasks for specialized handling. In some implementations, the planner's output might need further processing to assign tasks to different types of workers or to structure them properly. The router's responsibility is to take the high-level plan and break it down into concrete **actionable tasks for workers**, possibly adding any metadata or deciding which worker persona or tool to use for each task.

- **Conditional Routing:** If certain subtasks require special handling (for instance, some might be factual searches while others require calculation or code), the router decides which path or worker type should handle each. This is analogous to an if-else decision based on task type [8] [9]. For example, if one subtask is "Translate a document," the router might assign it to a translation worker vs. a web research worker for "Find latest papers on X."

- **Task Specification:** The router ensures each task is well-defined. It might transform a vague prompt into a more specific one for the worker. For instance, if the planner gave “History of AI”, the router could specify: “Gather a summary of AI development from 1950 to 2020, focusing on major milestones.”
- **Output:** A finalized list of tasks, each possibly annotated with how to handle it. This could be a JSON array of task objects ready for parallel execution. In many cases, the planner and router might be combined (the planner could output the task list directly). But separating them allows one agent to focus on *what* to do, and the other on *how/where* to do it – further adhering to single responsibility.

Worker (Parallel Researchers)

Worker agents carry out the actual research and data gathering for each task. They are the “map” phase in the Map-Reduce analogy: multiple workers run **in parallel**, each handling a different subtask allocated by the plan [1]. This massively speeds up the process and allows broad coverage, as opposed to a single agent doing one thing after another. Workers are typically using an LLM prompt geared towards analysis or information retrieval, possibly a smaller or specialized model optimized for these tasks [10].

- **Independent Execution:** Each worker takes a task (for example, “Summarize the history of AI from 1950–2020”) and executes it independently. They might perform tool actions like web searches, database queries, or just draw on their trained knowledge to generate an answer or summary for that subtask.
- **Parallelism:** Because tasks are independent (or made independent by the plan), workers can run concurrently. This design mimics the data-parallel nature of Map-Reduce: “map” jobs all process different chunks of the problem at the same time [2]. For instance, one worker might be summarizing AI history while another is compiling current AI technologies, simultaneously.
- **Output:** Each worker produces an output artifact – e.g. a text summary, data points, or analysis for its specific task. These outputs might be quite detailed. For example, the history task might yield several paragraphs covering AI’s evolution. At this stage, we may have *a lot* of information collected across all workers.

The system must now deal with these numerous outputs, which might be too large to simply concatenate for a final answer. That’s where the next role comes in.

Refiner (Intermediate Summarizer)

The **Refiner** acts as a mini-reducer for each worker’s output, compressing and polishing the information so it can be handled in the final synthesis. In essence, if each worker produced a chunk of text or findings, the refiner takes each chunk and **summarizes or distills it** into a more concise form. This mitigates the context overload issue by funneling a large volume of text down into key points. It’s a crucial step to ensure the final reducer isn’t overwhelmed by the combined size of all worker outputs.

- **Summarizing Outputs:** For each worker’s result, the refiner generates a condensed summary that retains the essential facts or insights. This could be done with prompts like “In 5 bullet points, summarize the key findings about [subtopic] from the above text.” By doing so, a 1000-word output might be reduced to a 200-word summary capturing all main ideas.
- **Highlighting Key Info:** The refiner can also standardize the format of outputs. For instance, it might ensure each summary includes references or specific data if needed, or it might extract just the most relevant names, dates, and conclusions. Essentially, it **filters noise and redundancy** out.

- **Single Responsibility:** Each refiner agent handles one worker output (this could technically be the same model used for all via a loop, but conceptually it's a batch of refining jobs). This separation means the prompt for refining can be specialized (e.g. concise summarization style), different from the prompt that the worker used for research.

After refinement, we have a set of **short, uniform summaries** – one per original subtask. These are far easier to fit into a single context for the final stage. We've transformed high-volume unstructured data into a manageable set of key insights, a step often recommended to handle LLM context limits (essentially a form of hierarchical summarization [5]).

Reducer (Integrator and Reporter)

The **Reducer** is the final synthesis agent that takes all the refined summaries and **integrates them into a coherent, comprehensive report or answer**. This is analogous to the reduce phase in Map-Reduce, where all the mapped data is aggregated to produce the final output. The reducer's prompt is typically aimed at high-level writing and reasoning: it must merge different threads of information, resolve any inconsistencies, and present the results in a structured format that addresses the original query.

- **Integration of Knowledge:** The reducer reads all the refined outputs (which should collectively contain the necessary info from each subtopic) and combines them. It looks for overarching themes or insights that emerge from the pieces. For example, if one summary was history of AI, another current tech, another future trends, the reducer might identify a narrative linking past, present, future – ensuring the final report flows logically.
- **Comprehensive Answering:** Because the planner's strategy covered the query from multiple angles, the reducer must ensure the final answer addresses all those angles. It is effectively writing the *final research report*. This could include an introduction (context of the query), sections or paragraphs for each subtopic (history, tech, future, etc.), and a conclusion. The result should be **detailed and holistic**, much like a thesis rather than a short Q&A snippet [11] .
- **Quality Composition:** The reducer can be configured to use a more capable or larger model, given its critical role in generating the final narrative [10] . The style should be polished and the content consolidated. At this stage, because it's working off summaries (not raw sources), it can focus on explanation and writing quality.

By the end of the reduce phase, we have a draft of the answer or report. It ideally stands on its own as a thorough response to the initial task. However, to truly ensure excellence and correctness, one more role is employed.

Reflector (Critic & Feedback Loop)

The **Reflector** (also known as a Critic or Evaluator) is a quality control agent that reviews the final output and decides if it needs improvement. This role implements an **evaluator-optimizer feedback loop**: the reflector evaluates the answer against certain criteria and can feed suggestions back to the Planner or Reducer for another pass [12] . This is analogous to an editor reviewing a draft in an iterative writing process, or in AI terms, having the model self-reflect and correct itself.

- **Evaluation:** The reflector reads the final report and checks it for completeness, accuracy, clarity, and alignment with the user's request. It might have a checklist or rubric (for example: "Does the report cover all subtopics? Is the information accurate and up-to-date? Is the explanation clear and logically structured?"). In some advanced setups, multiple reflectors could critique different aspects (factual correctness, coherence, grammar) [13].

- **Feedback Generation:** If shortcomings are found, the reflector formulates a feedback or an "improvement plan." For instance, it might notice missing details on a subtopic or an unclear section, and it could suggest: "Add more details about AI in the 1980s," or "Clarify the transition between current tech and future trends." This feedback is then passed to the appropriate agent for revision. Minor edits might be given to the Reducer to fix in the text; more substantial issues (like a missing subtopic) might be sent all the way back to the Planner to adjust the initial plan or to spawn a new worker task.
- **Iterative Refinement:** The system can loop through generate → evaluate → improve for a few iterations until the reflector is satisfied (or a max number of loops is reached to avoid infinite cycles) [14]. Each iteration should yield a better result. This patterned approach is inspired by recent techniques where one LLM output is critiqued by another and then improved accordingly, boosting quality in a structured manner.

Finally, when the reflector approves the output (or no major issues are detected), the process concludes. The end result is a well-researched, comprehensive answer that has been planned, expanded, distilled, and validated through multiple stages.

Example Workflow

To illustrate the above roles in action, consider a user asks: **"Give me a deep research report on the field of AI (Artificial Intelligence)."** The system would proceed as follows:

1. **Planner:** The planner agent analyzes the broad query and decides on a strategy. It might come up with a plan like: *split the report into three main parts – (A) History of AI, (B) Current AI Technologies, (C) Future trends and challenges in AI*. This plan (a list of three subtopics) is produced for the next stage.

2. **Router:** The router takes these subtopics and prepares specific tasks. For each subtopic, it might formulate a precise prompt or method. For example, for "History of AI", it might specify a task to *"Summarize the key milestones in AI from the 1950s to today, citing major events and figures."* All three subtopics are now well-defined tasks ready for execution. (If no special routing logic is needed, the plan is used as-is.)

3. **Workers (Discovery Phase):** Three worker agents are spawned in parallel, each handling one subtopic:

4. Worker 1 researches the **History of AI** and produces a detailed summary of AI developments by decade.
5. Worker 2 researches **Current AI Technologies** (like deep learning, neural networks, applications) and compiles an overview.

6. Worker 3 researches **Future AI Trends** (such as AGI prospects, ethical considerations, emerging techniques). Each worker might retrieve information from the web or its knowledge, possibly using tools, and then output a chunk of text with findings for its area.

7. **Refiners:** Now three refiner agents take each worker's output and condense it:

8. Refiner 1 reads the history output and produces a concise summary highlighting only the most seminal events (maybe bullet points of each era).
9. Refiner 2 condenses the current tech overview to key technologies and their significance.
10. Refiner 3 summarizes future trends, focusing on predictions and challenges. After this step, we have three summarized outputs (history summary, current tech summary, future trends summary), each much shorter and sharper than the original worker texts.
11. **Reducer (Reporting Phase):** The reducer agent takes the three refined summaries and integrates them. It might start with an introduction about AI, then have sections for history, current state, and future outlook, using the content from each summary to write those sections. It ensures the sections flow logically and references from one section can connect to another if needed (for instance, linking how historical evolution led to current tech). The outcome is a single, cohesive **research report on AI** that covers all parts of the query.
12. **Reflector (Quality Check):** The reflector now reviews the AI report. Suppose it finds that the section on "Current AI Technologies" is missing mention of a very recent development (say, a new breakthrough in AI from last month) or that the narrative between sections isn't smooth. The reflector would note these issues. It might loop back: perhaps it asks the worker stage to fetch info on that recent development, or simply ask the reducer to insert a clarifying paragraph. After the changes, the reflector re-evaluates. Once it deems the report comprehensive and well-written, it finalizes the answer.

Throughout this process, all intermediate data (plans, partial answers, summaries) are stored in a **shared state** or memory, so that any agent at a later stage can refer back if needed [6]. For example, if the reflector wants an addition, the system can recall what was generated in earlier steps to avoid inconsistency. The shared state also facilitates the feedback loop: the planner can adjust the plan knowing what was already done, etc.

In summary, this slim Map-Reduce-inspired workflow leverages multiple specialized LLM agents to conduct in-depth research. It moves from **planning (strategy)** to **execution (parallel fact-finding)** to **synthesis (summarization and integration)**, and finally to **validation (feedback refinement)**. By dividing the task, it avoids context overflow and plays to the strengths of each agent. This yields a more thorough and reliable result than a naive single-agent approach, as evidenced by similar orchestrated multi-agent systems in recent AI research [7] [15]. The design essentially functions like a team of researchers: a project manager (planner), various domain specialists (workers), editors (refiners/reducer), and a reviewer (reflector), all working together under an orchestrated process to produce a high-quality research output.

---

[1] [3] [4] [5] [6] [7] [10] [11] [15] Inside the Architecture of a Deep Research Agent - Egnyte Blog
https://www.egnyte.com/blog/post/inside-the-architecture-of-a-deep-research-agent/

[2] [8] [9] [12] [13] [14] 5 LLM Workflow Patterns for Production AI Systems
https://www.decodingai.com/p/stop-building-ai-agents-use-these