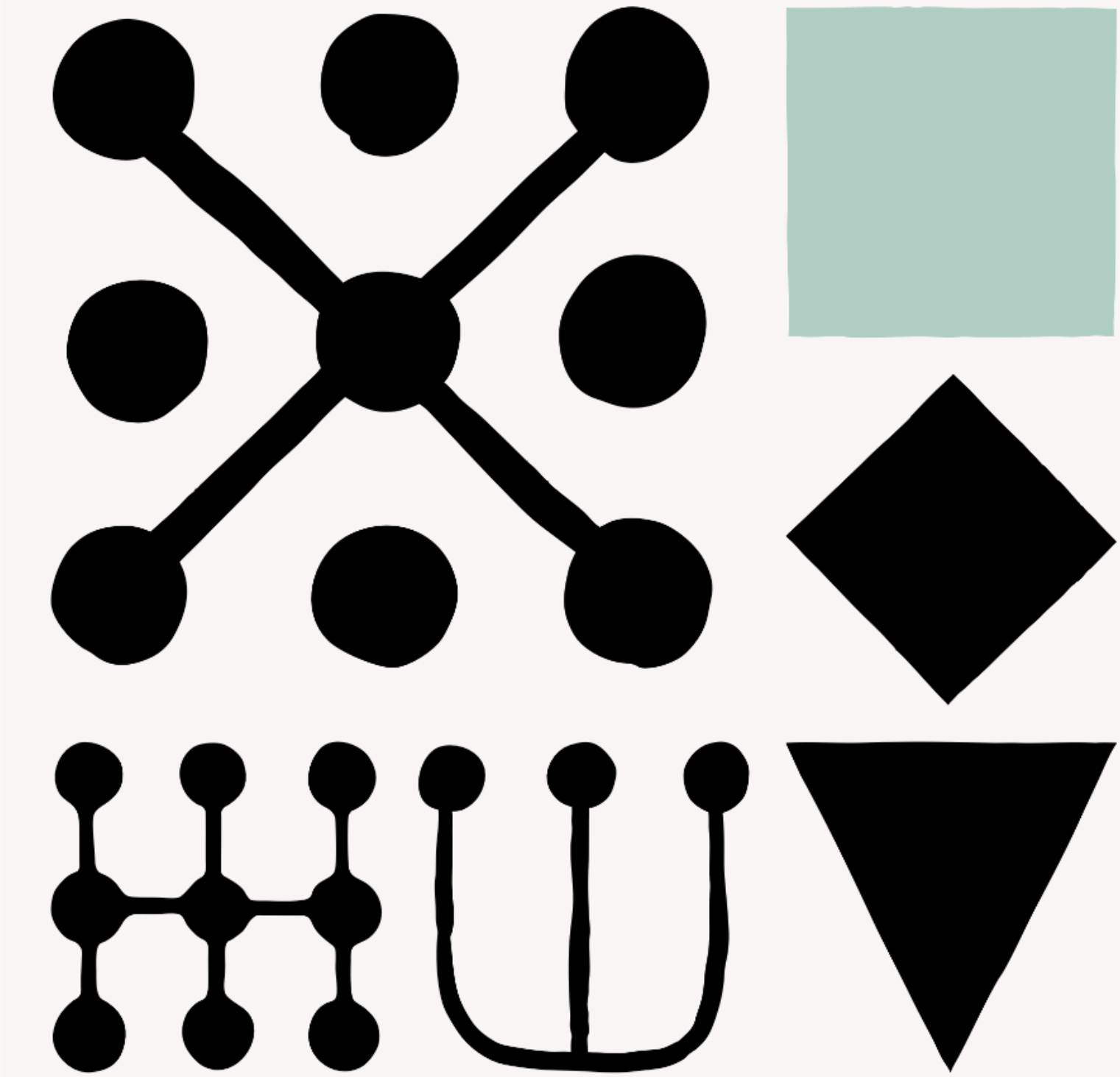
ANTHROP\C



How we built our multi-agent research system

Published Jun 13, 2025

Our Research feature uses multiple Claude agents to explore complex topics more effectively. We share the engineering challenges and the lessons we learned from building this system.

What’s new in Claude?

Claude now has Research capabilities that:

- Search the web, Google Workspace, and external tools.
- Use **multi-agent systems**—multiple LLMs coordinating autonomously.
- Complete complex research workflows with parallel reasoning.

🔍 The architecture helps Claude dynamically search, plan, and cite high-quality results.

This post breaks down the principles that worked for us—we hope you'll find them useful to apply when building your own multi-agent systems.

Why Multi-Agent Systems?

Research is dynamic: It involves open-ended, unpredictable paths—fixed sequences don’t work.
AI agents are adaptable: They can pivot based on intermediate findings, unlike linear pipelines.
Search is compression: Subagents explore different angles in parallel and distill key insights for the lead agent.
Separation of concerns: Each subagent handles specific tasks using its own tools and prompts—enabling deeper, independent investigation.

Scaling Performance

- Collective intelligence wins
- Breadth-first excellence
- Proven results

Why It Works

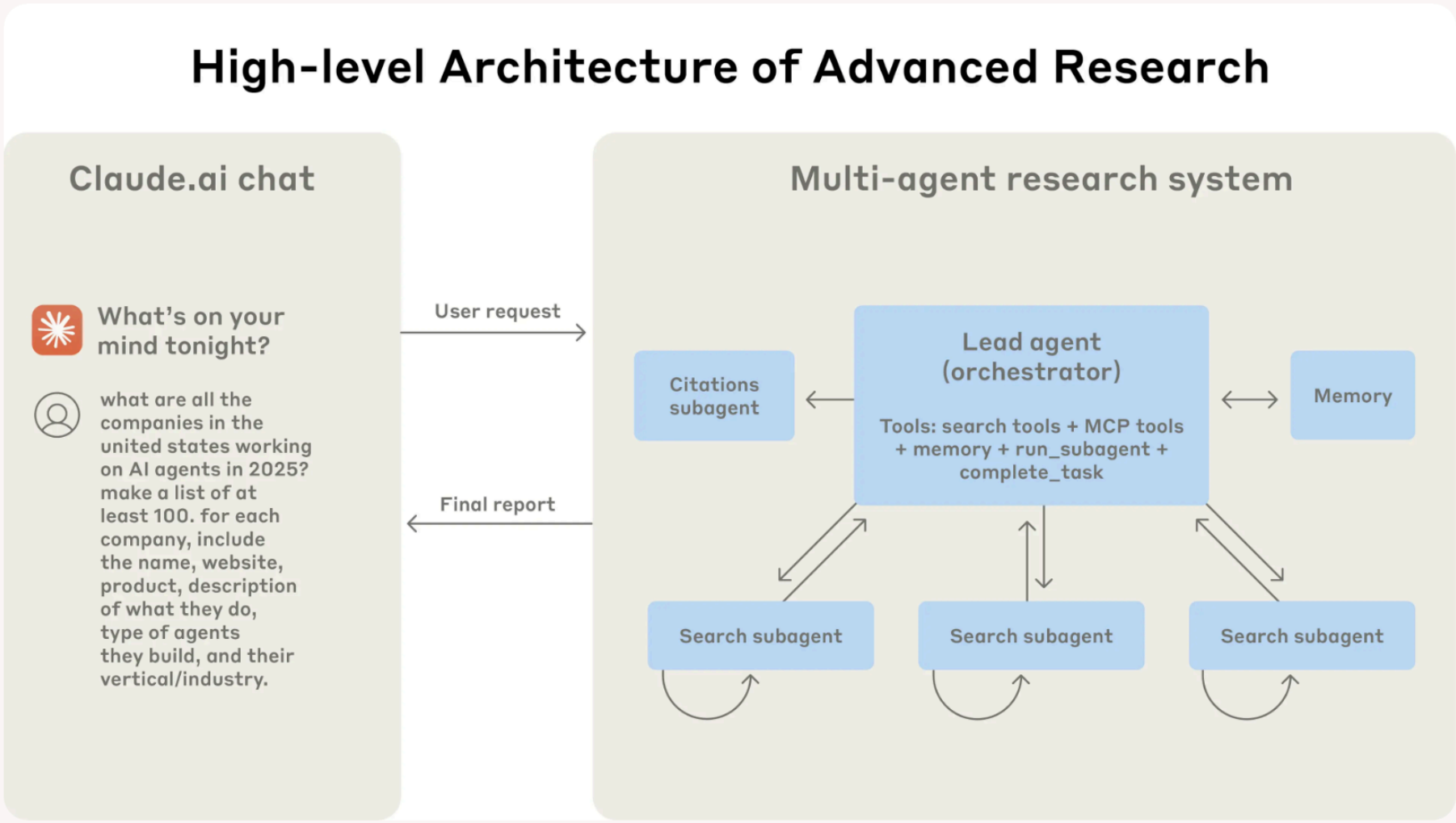
- Token usage is key
- Architecture matters
- Model choice + tool usage

Downsides & Constraints

- **High token consumption:** ~15× more tokens than standard chat interactions.
- **Best for high-value tasks:** Needs to justify higher compute cost.
- **Limited for some domains:**
  - Shared context requirements (e.g. in coding)
  - Real-time agent coordination is still a challenge

Architecture overview for Research

Our Research system uses a multi-agent architecture with an orchestrator-worker pattern, where a lead agent coordinates the process while delegating to specialized subagents that operate in parallel.



The multi-agent architecture in action: user queries flow through a lead agent that creates specialized subagents to search for different aspects in parallel.

When a user submits a query, the lead agent:

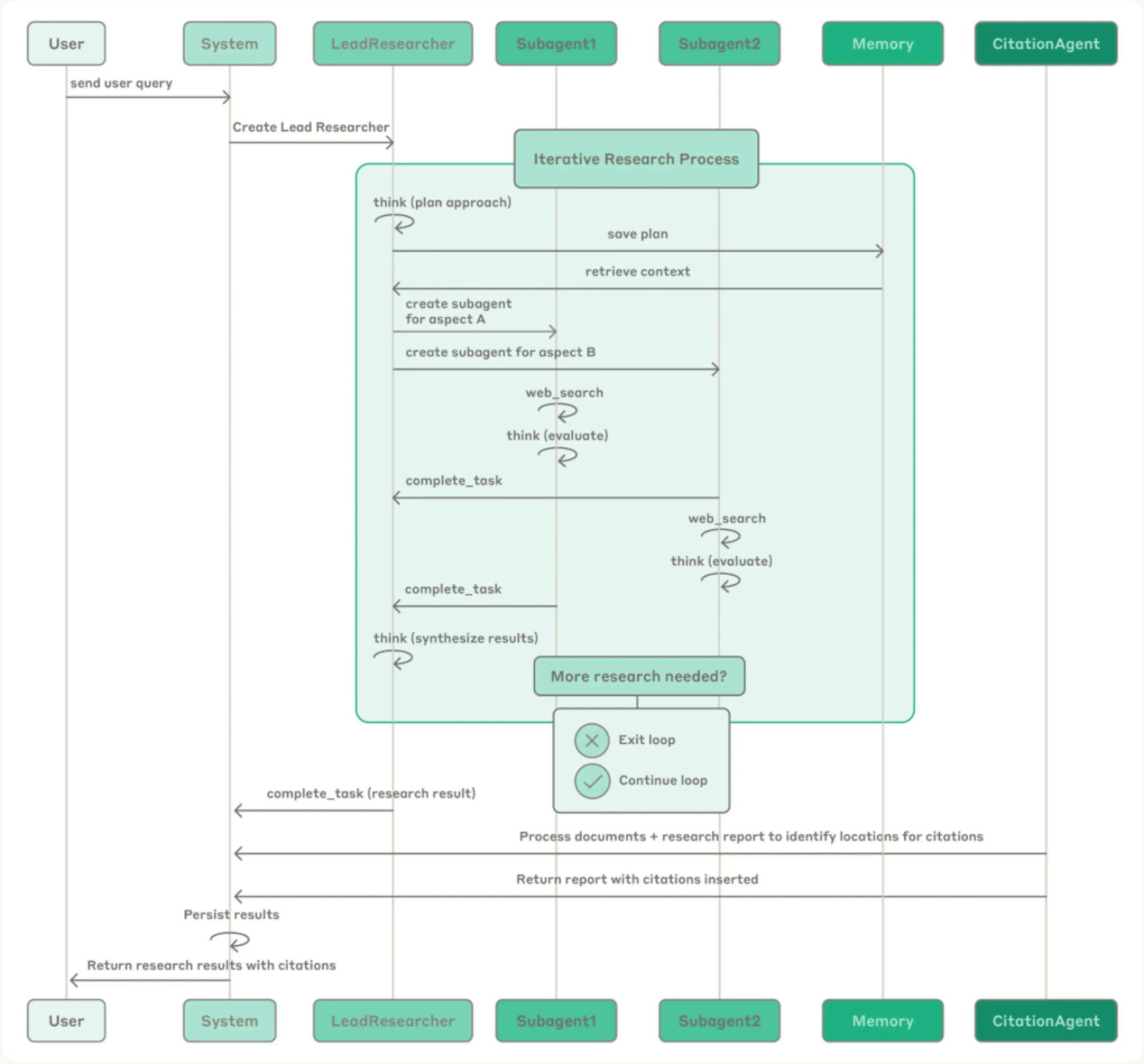
- Analyzes the query
- Develops a strategy
- Spawns **subagents** to explore different aspects in **parallel**

Subagents act as intelligent filters, using tools to gather and return insights to the lead agent. The system dynamically **compiles a comprehensive answer** from all subagents. Unlike **traditional Retrieval Augmented Generation (RAG)** which uses static chunk retrieval.

This system uses **multi-step, adaptive search,** allowing it to:

- Evolve based on new findings
- Produce **higher-quality answers**

Multi Agent System Process Diagram



Process diagram showing the complete workflow of our multi-agent Research system. When a user submits a query, the system creates a LeadResearcher agent that enters an iterative research process. The LeadResearcher begins by thinking through the approach and saving its plan to Memory to persist the context, since if the context window exceeds 200,000 tokens it will be truncated and it is important to retain the plan. It then creates specialized Subagents (two are shown here, but it can be any number) with specific research tasks. Each Subagent independently performs web searches, evaluates tool results using interleaved thinking, and returns findings to the LeadResearcher. The LeadResearcher synthesizes these results and decides whether more research is needed—if so, it can create additional subagents or refine its strategy. Once sufficient information is gathered, the system exits the research loop and passes all findings to a CitationAgent, which processes the documents and research report to identify specific locations for citations. This ensures all claims are properly attributed to their sources. The final research results, complete with citations, are then returned to the user.

Prompt engineering and evaluations for research agents

Multi-agent systems have key **differences from single-agent systems**, including a rapid growth in coordination complexity. **Early agents made errors** like **spawning 50 subagents** for simple queries, **scouring the web endlessly** for nonexistent sources, and **distracting each other** with excessive updates. Since each agent is steered by a prompt, **prompt engineering was our primary lever** for improving these behaviors. Below are some **principles we learned** for **prompting agents**:

1. Think like agents: Simulate agent behavior to identify failures and improve prompts with a better mental model.

2. Delegate clearly: Lead agent must assign well-defined tasks to avoid duplication and misalignment.

3. Scale by complexity: Use more agents/tools only for complex queries; avoid overkill on simple ones.

4. Design/select tools wisely: Good tool descriptions and usage heuristics are critical for success.

5. Let agents improve: Claude models can self-optimize prompts and tool descriptions, reducing task time.

6. Start broad, then narrow: Begin with general queries, refine as results guide the direction.

7. Guide thinking: Use extended thinking mode for better planning, reasoning, and adaptation.

8. Parallelize for speed: Spawn subagents and tool calls in parallel to massively cut research time.

9. Heuristics over rules: Encode expert strategies like depth vs. breadth, evaluation, and flexibility into prompts.

Effective evaluation of agents

Good evaluations are essential for building reliable AI applications, and agents are no different. **Traditional evaluations often assume** that the **AI follows the same steps each time**: given input X, the system should follow path Y to produce output Z. But **multi-agent systems don't work this way**. Even with identical starting points, agents might take completely different valid paths to reach their goal. We need **flexible evaluation methods** that judge whether agents achieved the **right outcomes** while also following a reasonable process.

Start small and early: Begin with a small set of real-world test queries to evaluate agent performance early in the development cycle. Even with just 15–20 queries, you can identify major improvements or regressions without waiting for large, time-consuming evaluation suites.

Use LLMs as judges: Leverage LLMs to assess agent outputs using structured rubrics that evaluate factual accuracy, citation correctness, completeness, and tool usage efficiency. This approach enables scalable evaluation of hundreds of outputs with high alignment to human judgment.

Don’t skip human testing: Human evaluators are essential for catching nuanced errors—like hallucinated facts or consistent preference for low-quality sources—that automated metrics may miss. Manual review provides critical insights, especially during edge-case handling and debugging.

Expect emergent behavior: In multi-agent setups, even minor changes to prompts or roles can drastically change how agents interact. Instead of prescribing rigid steps, design your prompts as flexible frameworks that support collaboration, delegation, and adaptive decision-making.

Key to success: The most effective multi-agent systems combine well-structured prompts, clearly described tools, robust heuristics, strong observability, and fast feedback cycles. Together, these elements ensure reliable behavior, faster iteration, and higher quality outcomes.

Production reliability and engineering challenges

In traditional software, a bug might break a feature, degrade performance, or cause outages. In agentic systems, minor changes cascade into large behavioral changes, which makes it remarkably difficult to write code for complex agents that must maintain state in a long-running process.

Stateful Agents with Compounding Errors: Agents often operate over long-running tasks and maintain state across many steps. Unlike traditional systems where a restart fixes things, errors mid-process can't be easily retried. Restarting from scratch is expensive and frustrating for users. Instead, robust systems must checkpoint agent progress, implement retry logic, and leverage the agent's own reasoning to detect and recover from tool failures.

Debugging Requires New Techniques: Agents are non-deterministic, meaning the same prompt can produce different behavior across runs. This makes debugging tough—problems like "agent didn't find obvious information" require deep insight into decision-making patterns, tool usage, and search effectiveness. The solution is to add full tracing and high-level observability, allowing developers to trace agent actions step-by-step and understand why it failed, without accessing private user content.

Safe Deployment is Complex: Agents interact with a complex mix of prompts, tools, and logic. Because agents can be midway through long processes, blindly updating code can break active workflows. To prevent this, teams use rainbow deployments—a strategy where new versions are rolled out gradually, ensuring that both old and new agents can run side by side without disruption.

Synchronous Execution Creates Bottlenecks: Most systems today execute subagents synchronously—the lead agent waits for all subagents to return before proceeding. This approach simplifies coordination but limits parallelism and prevents subagents from collaborating or adapting mid-task. Asynchronous execution, where agents work concurrently and spawn new agents as needed, promises major performance gains, but introduces challenges in maintaining state consistency, handling errors across branches, and merging results.

Conclusion

The "last mile" in AI agent development is often the hardest part, requiring extensive work to move from prototype to production.

Agentic systems are highly sensitive to small failures, which can compound and lead agents to pursue incorrect or unpredictable paths.

Traditional software bugs might cause minor issues, but in multi-agent systems, they can derail the entire workflow.

The transition from working code to production-ready systems is wider and more complex than expected.

Despite the complexity, multi-agent systems excel in open-ended research tasks, delivering value to users across diverse domains.

Users reported that **Claude helped uncover insights, solve complex problems, and save significant time** across business, healthcare, and technical areas.

Reliability at scale is achievable, but it requires:

- Strong prompt and tool design
- Detailed testing and engineering
- Robust operational practices
- Close collaboration between teams who understand agents deeply

Acknowlegements

Written by Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. This work reflects the collective efforts of several teams across Anthropic who made the Research feature possible. Special thanks go to the Anthropic apps engineering team, whose dedication brought this complex multi-agent system to production. We're also grateful to our early users for their excellent feedback.

From **ANTROPIC**

Interested in
more content like this?

Follow me :

OM NALINDE

linkedin.com/in/that-aum