Orchestrating Teams of Agents

Last modified: 2026-09-13 00:11:34 (PDT)

A single coding agent works one problem at a time. *Orchestration* is the step up from that: running several agents at once and coordinating their work. This chapter explains when orchestration is worth the added cost, describes what our lab already uses for it, evaluates three outside "agent orchestrator" projects that lab members have asked about, and surveys the general-purpose orchestration frameworks a lab could build its own agent systems on.

> ⚠ A fast-moving, opinionated snapshot
>
> As of early 2026, this space is changing weekly. The tools below are young, and the star counts, version numbers, and "experimental" labels quoted here were true when this chapter was written (August 2026) and will drift. The verdicts are the lab's current opinion, not settled fact. Re-check before acting on any of them.

1 When Orchestration Helps

Orchestration pays off when work splits into pieces that can run at the same time without waiting on each other. The strongest cases are:

- **Research and review**: several agents investigate different angles at once, then compare and challenge each other's findings.
- **Independent implementation**: each agent owns a separate module, file set, or task.
- **Adversarial verification**: agents test competing hypotheses in parallel and converge faster than one agent anchoring on its first guess.

Orchestration is not free. Every extra agent has its own context window and spends tokens independently, so cost grows with the number of agents, and coordination overhead grows too. For sequential work, edits to the same file, or tasks with many dependencies, a single agent is cheaper and simpler. The question for any orchestration tool is not "is it impressive?" but "what does it add to what we already do?"

2 What We Already Use

Our lab already orchestrates agents, using Claude Code plus the conventions in our `ai-config` repository. The pieces are:

- **Subagents**: a session spawns focused helper agents for search, review, or verification. Each has its own context window and reports its result back to the main session.
- **The `Workflow` fan-out tool**: a deterministic script that spreads work across many agents in parallel (for example, one reviewer per dimension of a pull request), with token budgets and verification stages built in.
- **Git worktrees**: isolated checkouts that let parallel sessions edit a repository without clobbering each other.
- **Hooks and permissions**: guardrails that gate risky actions, scope tool access, and enforce lab rules mechanically.
- **Reusable GitHub Actions workflows** (in `Morrison-Lab/gha`): the automated review-and-merge workflow for pull requests.

This baseline matters because it is the yardstick for everything below. A new orchestrator is useful to us only if it does something this stack does not already do well.

3 Claude Code Agent Teams

Agent Teams (Anthropic 2026) is the built-in Anthropic feature for coordinating several Claude Code sessions. One session acts as the **lead**, and it spawns **teammates**, each a full, independent Claude Code session with its own context window. Teammates coordinate through a shared task list and a mailbox, and, unlike subagents, they can message each other directly rather than only reporting back to the lead.

The intended uses match the strong cases in Section 1: parallel code review, competing-hypothesis debugging, new modules owned by different teammates, and changes that span several layers of a codebase. As of August 2026 the feature is **experimental** and off by default; it is enabled with the `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS` environment variable. It carries real limits: teammates do not survive session resumption, a session has exactly one team, teammates cannot spawn their own teammates, and token use is much higher than a single session.

> Useful to us? Yes, worth trying
>
> Agent Teams is a native extension of the tool we already live in, so it is the lowest-friction option here. It complements, rather than replaces, our subagents, `Workflow` fan-out, and worktree parallelism: teammates that talk to each other suit adversarial review and debugging, where our subagents (which only report back) are weaker. Treat it as a feature

to experiment with on research and review tasks, not a platform to adopt, and mind the token cost and the experimental limits.

4 Inflexa

Inflexa (Inflexa 2026) calls itself "the open-source orchestrator for computational biology." It is a local-first command-line tool that turns a plain-language biological question into code you can read and run, executes that code in a sandboxed Docker container, and records signed provenance for every step in a local database. It reads scientific literature, queries dozens of biological databases, and runs analyses in Python and R environments. It is model-agnostic: you bring your own key for Claude, an OpenAI-compatible endpoint, or a local model.

Inflexa is released under the permissive **Apache-2.0** license. Its open-source command-line tool is the full product, not a trial; the separate commercial offering only adds hosted infrastructure, team collaboration, and managed compute. As of August 2026 it is young but actively developed (around 30 stars), and its analysis "skill packs" are omics- and translational-medicine-focused (transcriptomics, proteomics, pharmacokinetics, drug repurposing, statistical modeling).

Useful to us? Yes, worth evaluating

Of the three outside tools, Inflexa is the most relevant to our actual research. Its priorities are ours: reproducible analysis in R and Python, provenance on every step, and no data leaving the machine. It is worth a real trial on a lab dataset, and its provenance and sandboxing design is a useful reference for our own reproducibility tooling. Two cautions: it is early, and its skill packs are aimed at omics rather than serology or infectious-disease epidemiology, so adopting it as a dependency now would be premature. Evaluate first; contribute a domain skill pack if the fit turns out to be good.

5 TORQCLAW

TORQCLAW (pilotwaffle 2026) is a "governed local and cloud AI-agent control plane." It is a TypeScript gateway, router, and console user interface wrapped around a forked Python execution engine, adding a governance layer between an agent's request and its action: approval gates, budget caps, spending receipts, and capability and path scoping over Model Context Protocol (MCP) tools. It runs local models through Ollama and falls back to a frontier model when needed.

As of August 2026 it is a single-author, early-stage project (a few stars, a completed "Phase 1" prototype). Critically, it ships **with no license file**, which under default copyright means all rights are reserved: we could not legally fork, vendor, or reuse its code even if we wanted to.

Useful to us? Not for adoption

TORQCLAW fails three ways for our purposes. It is unlicensed, so we cannot legally build on it. It is domain-agnostic agent *infrastructure*, not tooling for our research. And the governance ideas it centers on, approval gates, capability scoping, and cost limits, we already implement directly through Claude Code permissions and hooks, MCP scoping, and the `Workflow` tool's token budgets. At most, its design documents are worth skimming as a catalog of agent-governance patterns.

6 Multi-Agent Orchestration with Oh My OpenCode / Oh My OpenAgent

`code-yeongyu/oh-my-openagent` (originally published as **Oh My OpenCode** or `omo`, with community forks such as `opensoft/oh-my-opencode`) is an open-source multi-agent orchestration framework and plugin for AI coding agent harnesses (including OpenCode and OpenAI Codex CLI) with over 65,000 GitHub stars (measured 2026-09-01). Inspired by modular terminal configuration frameworks (such as Oh My Zsh), it expands single-agent coding into a specialized multi-agent system with automated model routing and background task execution.

Hub-and-spoke agent architecture

Rather than relying on a single generalist agent to perform all research, design, coding, and review steps, the framework implements a hub-and-spoke delegation model with specialized agent personas:

- **Sisyphus (Primary Orchestrator & Task Lead)**: Acts as the lead coordinator that breaks complex user requests into discrete work packages, delegates subtasks to specialized agents, and aggregates results into a coherent solution.
- **Prometheus (Strategic Planner)**: Interviews the user to clarify scope and ambiguities, formulating detailed implementation plans before code modifications begin.
- **Metis (Pre-Planning Gap Analyzer)**: Conducts pre-planning analysis alongside Prometheus to identify hidden requirements, ambiguities, and potential failure points before plans are finalized.
- **Momus (Plan Reviewer)**: Validates proposed execution plans against clarity, verification, and context criteria before execution.
- **Atlas (Plan Executor)**: Executes approved Prometheus plans step-by-step, managing task progress and session continuity across execution sessions.
- **Hephaestus (Autonomous Deep Worker)**: Specializes in autonomous architectural refactoring and complex multi-file debugging.

- **Oracle (Architecture & Debugging Consultant)**: Serves as a read-only architecture and deep debugging consultant, analyzing system design trade-offs and diagnosing subtle runtime failures.
- **Librarian (Documentation & Code Search)**: Conducts targeted external documentation lookups and code search to supply focused context without bloating the orchestrator's context window.
- **Explore (Codebase Navigation & Fast Grep)**: Specializes in fast pattern searches, codebase navigation, and file structure discovery.

Heterogeneous model routing

A central capability of the framework is decoupling agent roles from a single model provider. Developers can route distinct tasks to the model family best suited for each workload:

- **High-reasoning tiers** (such as Anthropic Claude Sonnet or Opus) for architectural planning and deep refactoring.
- **High-throughput models** (such as OpenAI GPT models) for rapid code generation and unit testing.
- **Large-context models** (such as Google Gemini) for extensive repository research and multi-file context indexing.

Operational capabilities

Dimension	Standard OpenCode	Oh My OpenCode / Oh My OpenAgent
Agent topology	Single sequential agent loop	Hub-and-spoke multi-agent team
Model routing	Single active model per session	Dynamic per-agent model assignment
Execution monitoring	Standard terminal output	Interactive `tmux`-backed session management
Extensibility	Individual plugins and MCPs	Curated bundle of tools, agents, and MCP integrations

7 Agent Orchestration Frameworks

The three projects above are finished tools. The projects in this section are *frameworks*: software libraries for building your own multi-agent system from agents, tools, and a control flow that you write in code. They matter to us for a different reason. Nobody in the lab

is asking to run `AutoGen`; the question is whether any of these libraries would let us build something our Claude Code stack cannot.

Every framework below shares one property worth stating up front. Each is a Python (or TypeScript, or .NET) library for writing an *application* that calls model APIs directly. Our workflow instead orchestrates a coding agent from the outside, through Claude Code's subagents, `Workflow` fan-out, hooks, and Git worktrees, and our research code is in R and Quarto. None of these frameworks has an R interface, so adopting one would add a Python application layer between us and the models, and that layer would need its own maintenance. That cost is the yardstick for every verdict here.

Star counts, release versions, and dates in this section were read from the GitHub API on 2026-09-09 and will drift.

7.1 Microsoft AutoGen

AutoGen (Microsoft 2026b) is Microsoft's original multi-agent framework, and the one issue #101 named first. Its current design is layered:

- `Core`: an event-driven, actor-style runtime for message-passing agents.
- `AgentChat`: the high-level API most users see, where agents are grouped into *teams* (round-robin group chats, model-selected speakers, swarm-style handoff) that pass a task around until a termination condition fires (Microsoft 2026a).
- `Extensions`: model clients and tools for outside services.
- `Studio`: a no-code web interface built on `AgentChat`.

It is Python-first (Python 3.10 or later), with a .NET port. The code is under the **MIT** license and the documentation under CC-BY-4.0 (Microsoft 2026c).

The decisive fact is on its own README (measured 2026-09-09): AutoGen is in **maintenance mode**. It "will not receive new features or enhancements and is community managed going forward," and new users are directed to the Microsoft Agent Framework below. The last release was `python-v0.7.5` on 2025-09-30, and the repository's last push was in April 2026, even though it still carries about 60,900 stars (Microsoft 2026c).

> **Useful to us? No**
>
> AutoGen is the best-known name on this list and is no longer where its own authors want new work to go. Its group-chat orchestration is what our subagents and `Workflow` fan-out already give us, without a Python application to maintain. If any Microsoft framework is worth watching, it is the successor.

7.2 Microsoft Agent Framework

The Microsoft Agent Framework (Microsoft 2026d) merges AutoGen with Semantic Kernel, Microsoft's earlier .NET-first library, into one production-oriented framework for Python and .NET, with a separate Go SDK. Orchestration is expressed as **graph-based workflows** with built-in sequential, concurrent, handoff, and group-collaboration patterns, plus checkpoints, streaming, human-in-the-loop steps, and `OpenTelemetry` tracing. Agents can also be declared in YAML. It is under the **MIT** license, has about 13,400 stars, and released `python-1.17.0` on 2026-09-03 (measured 2026-09-09).

> **Useful to us? Not now**
>
> It is the framework to name if someone asks "what replaced AutoGen?", and its workflow patterns are a clean catalog of the orchestration shapes that also appear in our `Workflow` tool. But its hosting story centers on Microsoft Foundry and Azure, which we do not use, and the general Python-layer cost applies.

7.3 LangGraph

LangGraph (LangChain 2026b) is LangChain Inc.'s "low-level orchestration framework and runtime for building, managing, and deploying long-running, stateful agents." Orchestration is a **graph**: you write nodes (a model call, a tool, or plain deterministic code), connect them with edges, including conditional ones, and share a typed state object between them. The runtime adds checkpoints, so a run can be paused, resumed, or replayed, interrupts for human-in-the-loop approval, and durable execution for long tasks. It is available in Python and JavaScript, can be used without the wider LangChain library, and integrates with LangSmith for tracing and hosted deployment (LangChain 2026a). The code is under the **MIT** license, has about 41,300 stars, and releases are tagged per package (the most recent, `sdk==0.4.4`, on 2026-08-27; measured 2026-09-09).

> **Useful to us? Worth knowing, not adopting**
>
> LangGraph is the most general and least opinionated framework here, and its checkpoint-and-resume model is the right design for a long analysis pipeline that must survive interruption. That is the one case where a lab member might reach for it: a durable, multi-step Python pipeline with human sign-off in the middle. For coordinating coding agents on a repository, which is what we actually do, it duplicates what Claude Code and the `Workflow` tool already provide.

7.4 CrewAI

CrewAI (CrewAI 2026b) organizes agents by **role**. A *crew* is a set of agents, each with a role, goal, and tools, working through a list of tasks either sequentially or under a manager agent (the hierarchical process), delegating to each other as they see fit (CrewAI 2026a). A *flow* is the newer, event-driven layer that manages state and decides when to run which crew. It is Python-only, under the **MIT** license, has about 58,300 stars, and released version `1.15.20` on 2026-09-04 (measured 2026-09-09). The open-source library sits under a commercial platform for deploying and monitoring crews.

> **i** Useful to us? No
>
> The role-playing metaphor is easy to start with and is the same thing our subagent definitions do (a reviewer role, a search role, a verification role), with less control over what each agent may touch. Nothing here is missing from our stack.

7.5 OpenAI Agents SDK

The OpenAI Agents SDK (OpenAI 2026) is a deliberately small framework built around a few primitives: agents, **handoff** between agents, agents used as tools, guardrails on input and output, sessions for conversation history, human-in-the-loop hooks, and tracing. Newer additions include sandbox agents that work inside a container over long tasks. Despite the name, it is provider-agnostic and works with any Chat Completions-compatible model and "100+ other LLMs". It is Python (3.10 or later), with a separate TypeScript package, under the **MIT** license, with about 29,300 stars and version `v0.22.2` released on 2026-09-09 (measured the same day).

> **i** Useful to us? No
>
> This is the OpenAI-side counterpart to Anthropic's Claude Agent SDK, and it would matter only if we built automation around Codex rather than Claude Code. Its handoff and guardrail design is worth reading as a reference, since it is the simplest statement of those ideas on this list.

7.6 Google Agent Development Kit

Google's Agent Development Kit (Google 2026) (ADK) is a "code-first Python framework for building, evaluating, and deploying" agents. Version 2.0 expresses orchestration two ways: a graph-based **workflow runtime** (routing, fan-out and fan-in, loops, retries, nested workflows, human-in-the-loop) and a **task API** for structured delegation between agents arranged in hierarchies. It is optimized for Gemini but model-agnostic, ships ports for Java, Kotlin, Go, and

TypeScript, and deploys to Cloud Run or Vertex AI Agent Engine. It is under the **Apache-2.0** license, has about 21,500 stars, and released `v2.8.0` on 2026-08-26 on a roughly two-week cadence (measured 2026-09-09).

> **Useful to us? No**
>
> ADK is the most complete of the vendor frameworks on paper, and the least relevant to us in practice: its strengths are Gemini and Google Cloud deployment, neither of which we use.

7.7 Others

Several more frameworks come up in the same conversations. None changes the verdicts above, so they get one line each (stars and dates measured 2026-09-09):

- `MetaGPT` (FoundationAgents 2026): the most-starred project on this list (about 70,300), simulating a software company of role agents; MIT, Python, but its last release was `v0.8.1` in April 2024 and its last push was January 2026.
- `LlamaIndex` (LlamaIndex 2026): document-centric agents and event-driven workflows; MIT, Python, about 52,100 stars, actively released.
- `smolagents` (Hugging Face 2026): Hugging Face's minimal library for agents that act by writing code; Apache-2.0, Python, about 29,300 stars.
- `Pydantic AI` (Pydantic 2026): typed, single-agent-first design from the `Pydantic` team; MIT, Python, about 19,800 stars, released daily.
- `CAMEL` (CAMEL-AI 2026): a research-oriented multi-agent framework; Apache-2.0, Python, about 17,700 stars.
- `VoltAgent`: a TypeScript framework, tracked separately in issue #45 of this site's repository.

8 Comparison

The table below places the three outside tools against our current stack. "Relevance to us" is the bottom line and follows directly from the rows above it.

Dimension	Claude Code + `ai-config` (ours)	Agent Teams	Inflexa	TORQCLAW
What it is	Our current agent stack	Multi-session teammates	Computational-biology orchestrator	Governed agent control plane

Dimension	Claude Code + `ai-config` (ours)	Agent Teams	Inflexa	TORQCLAW
Primary domain	General coding and research	General coding and research	Computational biology	General agent infrastructure
License	Reusable across our repos	Anthropic product	Apache-2.0 (permissive)	None (all rights reserved)
Maturity (2026)	In daily use	Experimental, off by default	Young, active	Early prototype, one author
Execution model	Subagents, `Workflow`, worktrees	Lead plus independent teammates	Sandboxed Docker, R and Python	TypeScript gateway, Python engine
Governance built in	Hooks, permissions, budgets	Inherits Claude Code permissions	Sandbox, signed provenance	Approval gates, budgets, receipts
Local-first	Yes	Yes	Yes	Yes (Ollama, cloud fallback)
Relevance to us	The baseline	Try it	Evaluate it	Pass; reference only

The frameworks in Section 7 do not fit this table, because they are libraries rather than tools: each would be a column only after we had built something with it. Measured against the same "Relevance to us" row, they land together at "reference only", with LangGraph the one to read first if a durable Python pipeline with human sign-off ever becomes a lab need.

9 Recommendation

For the lab, as of September 2026:

- **Try Agent Teams** on a research or review task where parallel, arguing agents would beat one agent working alone. It is native to our tooling and the cheapest to experiment with, as long as the token cost is watched.
- **Evaluate Inflexa** on a real dataset. It is the most domain-aligned option and openly licensed, and even if we do not adopt it, its provenance-and-sandbox design informs our own reproducibility work.
- **Pass on TORQCLAW** for adoption. It is unlicensed, off-domain, and centered on governance we already have.

- **Build nothing on the orchestration frameworks** in Section 7 until a concrete need appears that Claude Code cannot meet; read the LangGraph documentation as the reference design if one does.

None of these replaces our current Claude Code and `ai-config` practice. Agent Teams extends it, Inflexa is a candidate research tool alongside it, and TORQCLAW is, for now, only a reference.

References

Anthropic. 2026. *Orchestrate Teams of Claude Code Sessions*. Documentation. https://code.claude.com/docs/en/agent-teams.

CAMEL-AI. 2026. *CAMEL: Finding the Scaling Laws of Agents*. Software. https://github.com/camel-ai/camel.

CrewAI. 2026a. *CrewAI*. Software. https://github.com/crewAIInc/crewAI.

CrewAI. 2026b. *CrewAI Introduction*. Documentation. https://docs.crewai.com/en/introduction.

FoundationAgents. 2026. *MetaGPT: The Multi-Agent Framework*. Software. https://github.com/FoundationAgents/MetaGPT.

Google. 2026. *Agent Development Kit (ADK) for Python*. Software. https://github.com/google/adk-python.

Hugging Face. 2026. *Smolagents*. Software. https://github.com/huggingface/smolagents.

Inflexa. 2026. *Inflexa: The Open-Source Orchestrator for Computational Biology*. Software. https://github.com/inflexa-ai/inflexa.

LangChain. 2026a. *LangGraph*. Software. https://github.com/langchain-ai/langgraph.

LangChain. 2026b. *LangGraph Overview*. Documentation. https://docs.langchain.com/oss/python/langgraph/overview.

LlamaIndex. 2026. *LlamaIndex*. Software. https://github.com/run-llama/llama_index.

Microsoft. 2026a. *AutoGen AgentChat Tutorial: Teams*. Documentation. https://microsoft.github.io/autogen/stable/user-guide/agentchat-user-guide/tutorial/teams.html.

Microsoft. 2026b. *AutoGen Documentation*. Documentation. https://microsoft.github.io/autogen/stable/.

Microsoft. 2026c. *AutoGen: A Framework for Building AI Agents and Applications*. Software. https://github.com/microsoft/autogen.

Microsoft. 2026d. *Microsoft Agent Framework*. Software. https://github.com/microsoft/agent-framework.

OpenAI. 2026. *OpenAI Agents SDK*. Software. https://github.com/openai/openai-agents-python.

pilotwaffle. 2026. *TORQCLAW: A Governed Local and Cloud AI-Agent Control Plane*. Software. https://github.com/pilotwaffle/TORQCLAW.

Pydantic. 2026. *Pydantic AI*. Software. https://github.com/pydantic/pydantic-ai.