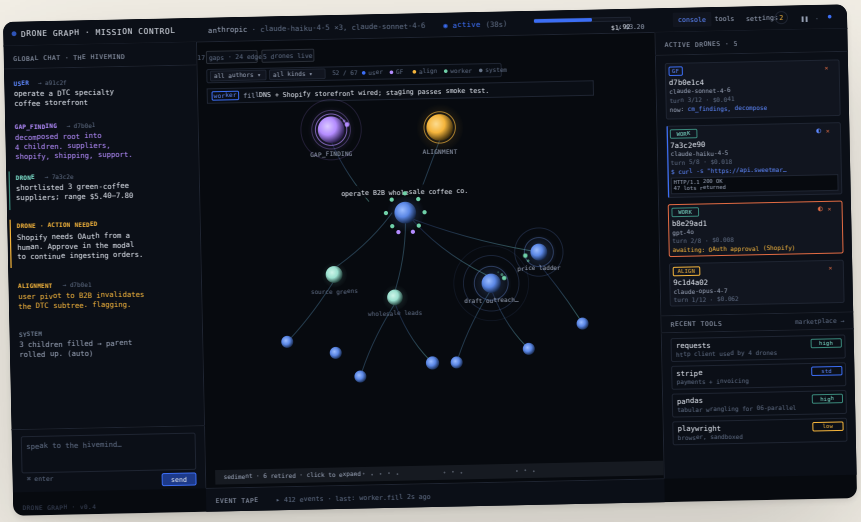
DRONE GRAPH · PAPERS · V1

Run your AI org *like a species.*

Two foundational notes on the architecture of AI swarms organized as a hivemind: drone theory, and the consolidated architecture overview.

Two papers · ~30 minutes



Mission Control — the dashboard for your swarm.

Dan Porder · Abhinav
2026-05-20 · MIT-licensed

github.com/drone-graph/drone-graph
dronegraph.dev

Drone theory

> *The theory: AI doesn't need specialisation, since frontier models have all the skills. Corporate division of labor and specialisation is an artefact of our limited time and that we are each born with different skills. AI is different.*

Similarly, corporate hierarchy is an artefact of people's need for dominance hierarchy, which AI doesn't have. AI can be more collaborative.

> *The hypothesis is: A more egalitarian structure for agent orchestration will be more effective than replicating typical corporate structure.A true agent swarm needs swarm intelligence — a hive mind / group mind. One shared consciousness across many units. Drones are individual interchangeable members of the system. In AI, their mind is a single shared collective mind.*

Examples from sci fi: Ultron, The Borg, The Geth, Replicators from Stargate, Formics/Buggers from Enders game, Zerg (Starcraft)

Architectural overview

PAPER 2 OF 2 · CONSOLIDATED ARCHITECTURE

Drone Graph — Architecture Overview

Mission control for the hivemind.

Core Loop

A drone is dispatched against a gap — preset or emergent. It loads its gap's intent + criteria + tool loadout, pulls what it needs from the collective mind, does the work, deposits findings, and dissolves. The process is infinite: Gap Finding always produces more gaps. Leaves (unfilled emergent gaps with no non-retired children) are the work pool for emergent worker drones.

Gaps

Gaps are the atomic unit of work — defined by absence, not assignment. Each gap carries:

- **Intent** — what must be true when the gap is satisfied
- **Criteria** — how to check that the intent is met
- **Tool loadout** — the explicit tool surface for any drone working this gap. Empty = inherit the default emergent loadout. Preset gaps and locked-down emergent gaps set this explicitly.
- **Tool suggestions** — recommended-but-not-preloaded tools the drone can pull in via `cm_request_tool` if it judges they're needed.
- **Context preload** — preset gaps declare a list of preloaders that the runtime renders at dispatch and injects into the drone's initial message (recent findings, leaves, tree shape). Saves the obvious-first-query turn.
- **Status** — `unfilled | filled | retired`.

There are two kinds of gaps. **Preset gaps** are persistent and always present — they are never filled, only continually acted on. Today the implementation mints two: Gap Finding and Alignment. Memory management, testing, and other presets are designed but not yet seeded. **Emergent gaps** are minted by Gap Finding's `decompose` and `create` verbs and are filled by drones as normal; the substrate auto-fills a parent when all its non-retired children are filled.

Gap Finding

Gap Finding is itself a preset gap. A drone dispatched against it edits the tree to keep it faithful to current signal: `decompose` to break a too-broad gap into children, `create` to add new top-level work,

`retire` to close off an invalidated subtree (children retire too), `reopen` to reconsider a filled gap whose intent wasn't actually met, `rewrite_intent` to reframe an unfilled gap when a signal explicitly invalidates its current framing AND the existing descendants stay coherent, and `noop` when nothing warrants a structural edit. Up to 5 verbs may be batched into one invocation so a dense signal (e.g. a user pivot) can be processed coherently in a single drone turn.

Drones

There is one drone class with one hivemind system prompt. The gap a drone is dispatched against — and that gap's `tool_loadout` — determines what the drone does. There is no per-role drone module. Preset behavior is expressed entirely through the preset gap's intent text and tool surface.

Drones can install new tools from the internet through their terminal and register them in the collective mind via `cm_register_tool`. Future drones discover those tools via `cm_list_tools` and pull them into their active set via `cm_request_tool`. Authored-skills (Claude Code style) and trust-tier ranking are designed but not yet implemented.

Signal Protocol

To avoid conflicting work, drones follow a signal protocol: check whether a package is already installed before installing; don't open a file another drone has open. Coordination is mechanical, not managerial.

The Terminal

The terminal is the persistent bash shell every emergent worker drone gets. It dies with the drone and respawns automatically if a single bad command crashes it (so an empty `{ }` block doesn't kill the whole worker). When a drone installs a package (e.g. Playwright), it should call `cm_register_tool` to add that capability to the registry as documentation — a future drone finds it via `cm_list_tools` and either re-installs from the recorded commands or runs it directly through `terminal_run`. Stale-tool pruning is designed but not yet implemented.

Collective Mind

The shared substrate. Persisted in Neo4j as `:Gap` + `:Finding` + `:Tool` nodes plus their relationships, with on-disk artefacts referenced by `Finding.artefact_paths`. Only what's actively needed is loaded into a drone's context (via the gap's `context_preload` and the drone's own `cm_*` queries); unused content is summarized and eventually pruned (to save disk).

- **Tool registry** — `:Tool` nodes. Builtins are mirrored at substrate init from a Python registry; installed tools are added at runtime by drones. Edges: `(:Tool)-[:USED_BY]->(:Gap)` (records who used what), `(:Tool)-[:DEPENDS_ON]->(:Tool)` (install ordering).
- **Findings** — short post-its written by drones. Each has author, kind, summary, optional `artefact_paths` to on-disk files, and `affected_gap_ids`.
- **Gaps** — the full surface of open work (preset and emergent), each with intent, criteria, tool policy, and status.
- **User uploads** — files and context provided by the human (referenced by path in findings, like any other artefact).

- **Skills** — runnable procedures any drone can load, download, or create. Designed; not yet implemented as a distinct concept beyond installed tools.

Everything in the collective mind is persisted, except what is explicitly pruned or summarized/compacted.

Human Role

Humans provide direction (initial goal, answering questions, uploading files) and handle a limited set of things the hivemind cannot do on its own — adding funds, performing authenticated actions (login, passwords), and similar. The list is small by design; everything else is the hivemind's problem.