Why mycoSwarm Was Born

February 8, 2026 · by Mark Bartlett

Download this post as PDF

I wanted Claude Code but couldn't justify $200/month. So I went looking for open alternatives. What I found was a trail of broken promises, security nightmares, and cloud-first assumptions masquerading as local AI tooling.

Six months later, I started building my own.

The Want

If you've seen Claude Code in action, you know the appeal. An AI that can read your codebase, run commands, edit files, iterate on errors. Not a chatbot you copy-paste to—an actual collaborator that operates in your environment.

Anthropic wants $200/month for that. Fair enough, it's genuinely useful. But I had a 3090 sitting under my desk and models that could run locally. Surely someone had built an open source version?

That search led me down a rabbit hole I'm still climbing out of.

The Drama

The first name I found was claudebot. Then moltbot. Then OpenClaw. Same project, different forks, accumulating drama like sediment.

The story, as far as I could piece together: original project gets popular, maintainers disagree, someone forks, community splits, accusations fly, another fork, more splits. The usual open source soap opera, except this one involved AI agents with shell access, which made the stakes higher and the drama pettier.

I didn't care about the personalities. I cared about finding something that worked.

The Horror

OpenClaw emerged as the “winner”—most stars, most active development, most plugins. So I cloned it and started reading.

440,000 lines of JavaScript.

Not a typo. Nearly half a million lines of TypeScript and JavaScript, sprawling across hundreds of files, with abstractions on abstractions and a configuration system that required a tutorial to understand. The kind of codebase where you open a file looking for one function and end up with thirty tabs open, still not sure how the request flows from input to output.

I’ve been writing code for decades. This wasn’t complexity born of hard problems. This was complexity born of inadequate pruning—features bolted onto features, never refactored, never simplified.

But I could have lived with messy code. What I couldn’t live with was the security.

Someone did an audit of ClawhubSkills—the plugin ecosystem for OpenClaw. The results were horrifying: 26% of submitted skills contained malicious code. Not subtle stuff. Prompt injection attacks. Data exfiltration. Remote code execution. Bad actors had figured out that people were giving these agents shell access and root trust, and they’d rushed to exploit it.

Meanwhile, 42,000 OpenClaw instances were running exposed to the internet with default configurations. No authentication. Full system access. A botnet waiting to happen.

The maintainers were too busy arguing about feature roadmaps to notice the house was on fire.

The Search

I needed something simpler. Something I could actually read.

Found nanobot from HKUDS (Hong Kong University). Completely different philosophy: 3,400 lines of Python. Clean architecture. Multi-agent support. Tool use. The kind of codebase where you can grep for a function and actually find it.

After the OpenClaw nightmare, it felt like fresh air. I set it up on my 3090 workstation, pointed it at my local Ollama instance running Qwen 2.5 14B, and started testing.

Ten minutes later, I found the bug.

The Bug

My queries were slow. Way too slow. A 14B model on a 3090 should respond in seconds, not minutes. I checked GPU utilization: nearly zero. The model wasn't running locally at all.

Dug into the logs. My "local" Qwen queries were being sent to Alibaba Dashscope—a Chinese cloud API.

The culprit was in `litellm_provider.py`, lines 107-112. Nanobot pattern-matches model names against known providers. See "qwen" in the model string? Must be Alibaba. Route to Dashscope. See "glm"? Must be Zhipu. Route to their cloud.

The `is_vllm` flag existed to override this behavior. But the routing check ran first. By the time the code checked whether you wanted local inference, it had already decided you wanted cloud.

A one-line fix: add `or model.startswith("ollama") or self.is_vllm` to the routing conditional. Ten minutes of debugging, five seconds of patching.

But the bug wasn't the point. The bug was a symptom.

The Pattern

Nanobot's maintainers didn't write that routing logic out of malice. They wrote it because cloud is the default assumption. Local inference is the edge case you accommodate after the fact.

The same pattern shows up everywhere:

LangChain defaults to OpenAI. Want local? Configure an alternative provider, hope the abstractions don't break.

LlamaIndex assumes you're calling an API. Local models are supported but secondary.

AutoGPT, MetaGPT, CrewAI—all designed around the assumption that inference happens somewhere else.

Even projects that claim to be "local first" treat local as "cloud, but on your machine." Same architecture, same assumptions, just with localhost instead of api.openai.com.

No one's building for a world where local is the starting point. Where your hardware is the primary compute resource, not a fallback. Where "no cloud dependency" isn't a feature flag but a core design principle.

The Question

What if we flipped the assumptions?

Local first. Not “local supported”—local as the default, with cloud as an optional upgrade if you want it.

Distributed by default. Not one machine running one model, but whatever hardware you have, working together.

Zero configuration. Nodes find each other. Capabilities get detected automatically. You don’t fill out YAML files to connect your own computers on your own network.

Sovereignty built in. Your prompts never leave your premises unless you explicitly send them somewhere. No API keys to manage, no rate limits to hit, no terms of service to violate.

What would that look like?

The Gap

I looked for existing projects that fit this description. Found pieces, but not the whole.

Exo does distributed inference across Apple Silicon devices. Clever work, genuinely impressive. But it’s inference only—no agent capabilities, no tool use, no task routing. Just raw token generation spread across machines.

Petals distributes large models across volunteers on the public internet. Cool concept, but I’m not sending my prompts through strangers’ computers, and the latency is brutal.

Ollama is excellent for single-machine local inference. No distribution story.

vLLM is built for servers with fat GPUs, not heterogeneous home networks.

The agent frameworks assume cloud. The inference frameworks don’t do agents. Nobody’s combined them in a way that treats “a gaming PC, two mini PCs, and a Raspberry Pi” as a unified compute resource.

So I Built It

I started writing mycoSwarm in February 2026.

The name comes from mycelium—the underground fungal networks that connect forests. No center, no hierarchy. Intelligence emerging from simple nodes in connection.

Week one: hardware detection. Every node knows what it has—GPU model, VRAM, CPU cores, available RAM, what Ollama models are loaded. A 3090 workstation identifies itself as "executive" tier. A ThinkCentre M710Q identifies as "light" tier. They know their capabilities without being told.

Week two: mDNS discovery. Nodes announce themselves on the local network and listen for others. No configuration, no IP addresses to enter, no central registry. Start the daemon on two machines and they find each other in under a second.

Week three: API layer. Every node exposes the same endpoints—health, status, peers, tasks. Any node can query any other node. The swarm becomes observable.

Current state: two nodes on my desk talking to each other. The 3090 knows it can run 32B models. The ThinkCentre knows it can handle embeddings and lightweight inference. They see each other. They know each other's capabilities.

Next: actual task routing. Type a prompt on the ThinkCentre, have it route to the 3090 for inference, return the result. The first real thought crossing the mycelium.

Why This Matters

I didn't start this project because I thought the world needed another AI framework. The world has plenty of those.

I started it because I wanted to use my own hardware to run AI agents without sending my data through cloud APIs, without trusting plugin ecosystems that are 26% malicious, without wrestling with half a million lines of JavaScript, without my "local" queries being silently rerouted to servers in China.

That shouldn't be a hard thing to want. But finding software that actually delivers it—without compromises, without asterisks, without "just configure these twelve settings"—turned out to be impossible.

So now I'm building it.

Maybe it works. Maybe the coordination overhead kills the theoretical gains. Maybe I discover why nobody else has built this. That's what the project is for: to find out.

The code is on GitHub. MIT license. No enterprise tier coming. No cloud service waiting in the wings.

If you've got old hardware sitting in a drawer and you're curious whether it can be part of something larger, come watch. Or better yet, come build.

📚 **Related:** What Open Source Was Supposed to Be · Best Local Models for OpenClaw · Multi-GPU Local AI

mycoSwarm: GitHub
