mycoSwarm vs Exo vs Petals vs Nanobot: What's Actually Different

February 8, 2026 · by Mark Bartlett

Download this guide as PDF

Every project in the local AI space claims some combination of "distributed," "local," or "private." Most of them are lying—or at least being selective with the truth.

The real question isn't whether software can run locally. It's whether you control the routing. When you type a prompt, who decides where it goes? You, or the software?

That's the lens for comparing these projects.

Exo: Distributed, Apple-Only, Inference-Only

Exo does one thing well: shard large models across multiple Apple Silicon devices. Got an M2 MacBook and an M3 Mac Mini? Exo lets them pool their unified memory to run models neither could handle alone.

What it does:

- Splits model layers across devices
- Automatic peer discovery on local network
- Works with Llama, Mistral, other popular architectures
- Genuinely impressive engineering

What it doesn't:

- No agent capabilities—inference only
- Apple Silicon only—your 3090 is useless here
- No tool use, no file access, no shell commands
- No task routing beyond "run this model"

Who controls routing: You do, but your only option is "these specific Macs running this specific model." No heterogeneous hardware, no fallback to cloud, no choices at all really—just one path.

Best for: People with multiple Macs who want to run larger models than any single machine allows.

Petals: Distributed Across Strangers

Petals takes a different approach: distribute inference across volunteers on the public internet. Anyone can contribute GPU capacity; anyone can run queries.

What it does:

- Runs models too large for any single volunteer
- BitTorrent-style distributed inference
- Supports big models (BLOOM, Llama)
- Free to use if you're willing to wait

What it doesn't:

- Privacy—your prompts traverse strangers' machines
- Speed—latency is brutal, often 10-30+ seconds per response
- Reliability—volunteers come and go
- Agents or tool use

Who controls routing: Nobody and everybody. Your prompts go wherever there's capacity. You have no idea whose GPU is processing your data. Hope you weren't working on anything sensitive.

Best for: Researchers who need occasional access to very large models and don't care about latency or privacy.

Nanobot: Clean Code, Cloud Assumptions

Nanobot is what got me started down this path. A refreshingly small codebase—3,400 lines of Python versus OpenClaw's 440,000 lines of JavaScript. Multi-agent support, tool use, clean architecture.

Then I discovered the routing bug.

What it does:

- Multi-agent orchestration
- Tool use (file access, shell, web search)

- Spawns sub-agents for complex tasks
- Actually readable code

What it doesn't:

- Local-first routing—cloud is the default
- Distributed inference across machines
- Automatic hardware detection
- Zero-config peer discovery

Who controls routing: The code does, and it assumes cloud. Model name contains “qwen”? Route to Alibaba Dashscope. Contains “glm”? Route to Zhipu. The `is_vllm` flag exists but the cloud check runs first. Your “local” queries can end up in China without any indication.

You can patch it. I did, in five seconds. But the architecture reveals the assumption: cloud is normal, local is the exception you configure.

Best for: Developers who want a lightweight agent framework and don't mind reading the routing code carefully.

OpenClaw: Features at Any Cost

For completeness, OpenClaw deserves mention. It's the 800-pound gorilla of open source AI agents—most GitHub stars, most plugins, most users.

What it does:

- Full agent capabilities
- Extensive tool ecosystem
- Active development
- Runs locally (technically)

What it doesn't:

- Stay secure—26% of ClawhubSkills were malicious
- Stay simple—440,000 lines of TypeScript
- Respect your time—configuration alone requires a tutorial
- Distributed inference

Who controls routing: You do, if you can find the right config file among hundreds. The real problem isn't routing—it's that the attack surface is so large that controlling routing doesn't help when the plugin ecosystem is compromised.

Best for: People who need every feature and have time to audit their entire stack.

The Gap

Here's what doesn't exist:

Feature	Exo	Petals	Nanobot	OpenClaw
Local-first (no cloud default)	✓	✗	✗	✓
Distributed inference	✓	✓	✗	✗
Agent capabilities	✗	✗	✓	✓
Tool use	✗	✗	✓	✓
Heterogeneous hardware	✗	✓	✗	✗
Zero-config discovery	✓	✗	✗	✗
You control all routing	✓	✗	✗	~
Readable codebase	✓	✓	✓	✗

No project checks every box. You have to choose: agents without distribution, or distribution without agents. Local without heterogeneous hardware, or heterogeneous without privacy.

What mycoSwarm Is Building

mycoSwarm aims to fill that gap:

Feature	mycoSwarm
Local-first (no cloud default)	✓
Distributed inference	✓ (in progress)
Agent capabilities	✓ (planned)

Feature	mycoSwarm
Tool use	✓ (planned)
Heterogeneous hardware	✓
Zero-config discovery	✓
You control all routing	✓
Readable codebase	✓ (~2,000 lines)

The core principle: **you control where every prompt goes.** Not the software. Not pattern-matching on model names. Not volunteers on the internet. You.

A task arrives. Your orchestrator checks capabilities across the swarm. The 3090 workstation can run 32B models. The ThinkCentre can handle embeddings. The Raspberry Pi can coordinate. You've defined the routing rules—or accepted sensible defaults that never leave your network.

Nothing goes anywhere you didn't explicitly allow.

Three hours of coding got us here: hardware detection, capability classification, mDNS discovery, API layer, two nodes finding each other in under a second. That's what happens when you start simple and stay simple.

Who Should Use What

Use Exo if: You have multiple Macs and want to run larger models. It does that one thing well.

Use Petals if: You're a researcher who needs occasional access to huge models, doesn't care about latency, and isn't working with sensitive data.

Use Nanobot if: You want a lightweight agent framework, understand the routing assumptions, and are comfortable patching code.

Use OpenClaw if: You need maximum features, have time for security audits, and don't mind 440K lines of JavaScript.

Use mycoSwarm if: You want agents + distribution + total routing control on whatever hardware you already own—and you're willing to use early-stage software.

We're not pretending mycoSwarm is production-ready. The discovery works. The capability detection works. The API layer works. Actual distributed inference is next. But the architecture is right: local-first, distributed by default, routing under your control.

That's the difference that matters.

📚 **Related:** What Open Source Was Supposed to Be · Why mycoSwarm Was Born · Multi-GPU Local AI · Planning Tool

mycoSwarm: GitHub

Source: https://insiderllm.com/guides/mycoswarm-vs-exo-petals-nanobot/