From 178 Seconds to 19: How a WiFi Laptop Borrowed a GPU's Brain

February 12, 2026 · by Mark Bartlett

Download this post as PDF

📚 **More on this topic:** mycoSwarm vs Exo vs Petals vs Nanobot · Multi-GPU Local AI · Ollama vs LM Studio · How Much Does Local AI Cost? · Planning Tool

I have a ThinkPad on the couch. No discrete GPU. Intel integrated graphics. The kind of laptop where running `ollama run` on anything bigger than a 3B model means you go make coffee. Possibly lunch.

I also have a workstation two rooms away with an RTX 3090. Twenty-four gigs of VRAM. Sitting there, rendering nothing, fans barely spinning.

The question that started mycoSwarm's GPU sharing experiment: what if the laptop could just use that 3090?

---

178 Seconds of Waiting

First I established a baseline. The ThinkPad, on its own, running a 7B model via Ollama. CPU inference. No GPU acceleration because there's no GPU to accelerate with.

I asked it to analyze a technical document and generate a structured summary. Nothing exotic. The kind of query you'd run fifty times a day if the responses came back fast enough.

178 seconds. Almost three minutes of the CPU grinding at 100%, fans screaming, the laptop getting hot enough to be uncomfortable on my legs. The response was fine. Getting there was miserable.

This is the reality for anyone running local AI on a laptop without a dedicated GPU. The models work. The hardware just isn't built for it. And buying a separate GPU-equipped machine feels like overkill when there's already one sitting in the other room.

The Setup: Fifteen Minutes, Two Machines, Zero Port Forwarding

Here's what I did. The whole thing took about fifteen minutes, and most of that was waiting for Tailscale to install.

On the workstation (RTX 3090):

1. Installed Tailscale and logged in. One command on Linux:

```
curl -fsSL https://tailscale.com/install.sh | sh
sudo tailscale up
```

1. Told Ollama to listen on all network interfaces instead of just localhost:

```
# Edit the Ollama service file
sudo systemctl edit ollama.service

# Add this under [Service]:
Environment="OLLAMA_HOST=0.0.0.0"

# Restart
sudo systemctl daemon-reload
sudo systemctl restart ollama
```

That's it for the workstation. Ollama is now accepting connections from the Tailscale network on port 11434.

On the ThinkPad (no GPU):

1. Installed Tailscale, logged in with the same account. The two machines found each other instantly. Tailscale gave the workstation an IP like `100.64.x.x`.
2. Pointed the laptop at the workstation's Ollama instance:

```
export OLLAMA_HOST=http://100.64.x.x:11434
```

That's the whole setup. No port forwarding. No firewall rules. No dynamic DNS. No VPN configuration beyond "install and log in." Tailscale handles all of it because it's a WireGuard

mesh under the hood. The laptop and workstation talk directly to each other, encrypted, peer-to-peer, even though they're on different subnets (the laptop on WiFi, the workstation hardwired to the router).

---

19 Seconds

Same model. Same prompt. Same document analysis query. I ran it from the ThinkPad's terminal expecting some improvement.

19 seconds.

I sat there and blinked at the screen. Ran it again to make sure something weird hadn't happened. 21 seconds. Again: 18 seconds. Consistent.

The laptop's CPU barely registered the workload. Because the laptop wasn't doing inference anymore. It was sending a prompt over the Tailscale tunnel, the 3090 was doing all the matrix math, and the tokens were streaming back over the wire. The ThinkPad had become a thin client for the workstation's GPU.

	CPU (ThinkPad)	Remote GPU (RTX 3090)
Time	178 seconds	19 seconds
Speedup	baseline	9.4x faster
CPU usage	100% (all cores)	~2% (just network I/O)
Fan noise	hairdryer	silent
Laptop temperature	uncomfortably hot	room temperature

The 9.4x speedup was the number. But honestly, the thing that hit me was the fan noise. Or rather the lack of it. The laptop was cool and quiet, and responses were coming back faster than they ever had running locally. It felt wrong, in a good way, like getting something for free.

What's Actually Happening Under the Hood

None of this is complicated. It's plumbing, wired up in the right order.

Tailscale creates a WireGuard mesh between your devices. Not a hub-and-spoke VPN where traffic routes through a central server. A mesh, where the laptop and the workstation talk directly.

Even when they're on different subnets, different VLANs, different physical network segments. Tailscale punches through NATs using a technique that sends packets from both sides simultaneously, so each NAT sees outbound traffic and opens the return path. When that works (it almost always does), the connection is direct. When it doesn't, Tailscale falls back to DERP relay servers, which forward encrypted WireGuard packets without decrypting them.

Ollama exposes a dead-simple HTTP API on port 11434. When you run `ollama run llama3` locally, the CLI is just making HTTP requests to `http://localhost:11434/api/generate`. Change that target from localhost to a Tailscale IP, and the CLI doesn't know the difference. The API call goes over the tunnel, the remote Ollama instance loads the model into its local GPU VRAM, runs inference, and streams the response back.

The laptop is doing nothing computationally expensive. It sends a JSON payload (your prompt), receives a stream of JSON payloads (the tokens), and displays them. The network overhead is negligible because prompts are small (a few kilobytes) and token responses stream one at a time. Even over WiFi, the latency added by the network is a rounding error compared to the inference time saved by using a GPU instead of a CPU.

That's why the speedup is so dramatic. It's not "GPU is a little faster than CPU." It's "GPU does this work in a fundamentally different way, and network latency is irrelevant at this timescale."

Why This Matters More Than It Looks

The "one machine, one GPU, one user" model is how everyone runs local AI right now. You install Ollama on your desktop, pull a model, and you're the only person who uses that hardware. If you move to your laptop, you either run inference on the laptop's weaker hardware or you don't use AI at all.

This demo breaks that assumption.

One good GPU can serve every device in the house. Your partner's laptop, your tablet via Open WebUI in a browser, your kid's homework machine. Nobody needs their own GPU. Scale that to a small office: one workstation with 24GB VRAM, every employee's laptop uses it. For a five-person team, that's the difference between $3,500 in GPUs and $700.

It gets wilder. A friend across town has a GPU they're not using right now. You add them to your Tailscale network (takes 30 seconds) and borrow their GPU from your couch. This works over the internet, not just LAN. I tested it with a machine on a completely different network and the experience was identical.

This is what mycoSwarm is building toward. Not just "point a laptop at a GPU." Automatic capability detection across the swarm. The orchestrator sees the 3090 workstation, the ThinkCentre with integrated graphics, the Raspberry Pi that can coordinate. It routes each query to the right hardware. You don't manually set `OLLAMA_HOST` because the system figures out which node should handle which model.

We're not there yet. The Tailscale + Ollama demo is manual. You configure the endpoint yourself. But it proves the core premise: GPU time is a shareable resource, not a per-device requirement. The plumbing works. Now it's about automation.

What This Doesn't Solve

I'd be lying if I said this was a complete distributed AI solution. It's not.

This is endpoint routing, not model sharding. The 3090 runs the entire model. If you need a model that doesn't fit in 24GB of VRAM, this doesn't help. You still need a bigger GPU or a system like Exo that splits models across devices.

Latency matters over long distances. On my LAN (and over Tailscale with direct peering), the added latency was invisible. Over a slow internet connection with DERP relay fallback, you'd feel it. This works best when the machines are physically nearby or on fast connections.

Security is on you. Exposing Ollama on `0.0.0.0` means anything on that network can hit it. Tailscale's access controls help here (you can restrict which devices can talk to each other), but you need to think about this. Don't expose Ollama on a public interface without Tailscale or a firewall in front of it.

Concurrent users will slow things down. The 3090 can handle one user's inference requests without breaking a sweat. Five people hammering it simultaneously will degrade performance for everyone. There's no built-in queueing or load balancing in Ollama itself. That's something mycoSwarm's orchestration layer will need to handle.

Try It Yourself

If you have two machines and fifteen minutes, you can replicate this.

What you need:

- One machine with a GPU and Ollama installed

- One machine without a GPU (or with a weaker one)
- A Tailscale account (free for personal use, up to 100 devices)

Steps:

1. Install Tailscale on both machines, log in with the same account
2. On the GPU machine, set `OLLAMA_HOST=0.0.0.0` and restart Ollama
3. On the other machine, run: `OLLAMA_HOST=http://<tailscale-ip>:11434 ollama run llama3.2`
4. Watch inference happen on someone else's GPU

If you want a web interface, install Open WebUI on either machine and point its Ollama URL at the GPU machine's Tailscale IP. Now you have a ChatGPT-style interface running on your laptop, powered by a GPU across the house.

What Comes Next

The 19-second moment was the proof. The laptop borrowed a brain and the brain didn't mind. No distributed systems PhD required. Two machines, a mesh VPN, one environment variable.

mycoSwarm takes this further. Instead of manually setting one environment variable, the swarm discovers every capable node on your network, catalogs their hardware (this node has a 3090, that one has 16GB unified memory, this Pi handles coordination), and routes queries to the right place automatically.

But even without mycoSwarm, the basic demo works right now. You probably have the hardware already. If you have a desktop with a GPU and a laptop without one, this is fifteen minutes of setup for a permanent upgrade to every device on your network.

The GPU doesn't care which machine asked the question. It just answers.

📚 **Related:** mycoSwarm vs Exo vs Petals vs Nanobot · Multi-GPU Setups: Worth It? · Open WebUI Setup Guide

mycoSwarm: GitHub

Source: https://insiderllm.com/blog/mycoswarm-wifi-laptop-borrowed-gpu/