Nvidia PAIR Bought Me 7 Percent. My Own Router Cost Me Half.

September 4, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** PAIR, Nvidia's new Personal AI Router — free software that spreads AI requests across the computers already on your home network — got 2x on three devices measured against their weakest. I got 7 percent on two against my strongest. My own router broke the same rule worse: every request to the slow card, half the speed of no router.

📚 **More on this topic:** Multi-GPU Setups: Worth It? · mycoSwarm vs Exo vs Petals vs Nanobot · Model Routing for Local AI · InsiderLLM Benchmarks

You have a good GPU in one machine and a slower one in another, and your agents queue behind the single card while the other one sits idle. PAIR, Nvidia's new Personal AI Router — free software that spreads AI requests across the computers already on your home network — is Nvidia's answer to exactly that. Nvidia announced it at IFA on 3 September; the beta I installed is v0.1.1, and I had it paired across my two bench machines by lunchtime.

Should you install it

Two or more similar machines: yes. Pairing took 18 seconds, it needed no root, it adopted the Ollama I already had on each box, and it routed correctly on every run.

One fast machine and one much slower one: only if the slow one is less than N times slower. PAIR splits requests evenly across N nodes, so a second node helps only if it is under twice as slow as the first. An RTX 3060 next to an RTX 3090 sits right at that line — 4.4 s against 2.35 s per request — and PAIR bought 7 percent. Before an engine upgrade on the 3060 it was 5.0 s, and PAIR was 2 percent slower than the 3090 on its own.

Bursts deeper than 120 seconds per node: not yet. PAIR's proxy gives up on any request that waits 120 s for the engine to start answering, and Ollama answers one at a time. Twenty 27B requests through PAIR: ten came back and ten failed with a 502. The same twenty straight to Ollama: all twenty.

The rule

PAIR's scheduler counts jobs, and nothing else. When twenty requests arrive at once it deals them out like cards — on my two nodes, exactly ten to the 3090 and ten to the 3060, on every one of nine PAIR runs across three sessions.

Ten requests on the 3060 at 4.4 s is 44 s, and that is your wall clock, because the 3090 finished its ten at the 23-second mark and sat idle. Twenty on the 3090 alone is 47 s. So the rule: **with even splitting, the slowest node must be less than N times slower than the fastest, or the cluster loses to the fastest machine alone.**

A load-aware split is the same sum the other way. On the round numbers from the first session, 2.5 s and 5.0 s, thirteen requests to the 3090 and seven to the 3060 gives a 35 s wall clock, or **1.40x** — exactly where I had pre-registered the expectation. On the later service times it is closer to 1.5x.

None of this is a discovery. Nvidia's known issues page has a section titled "Scheduling Only Counts Jobs" and says plainly that on a mixed cluster you should "expect work to land on a slower node about as often as a faster one."

If your hardware is mixed

Route by model name. PAIR's eligibility gate is a hard filter: a node only gets a request for a model it holds. Pull the small models on both cards and the big ones on the fast card only, and the slow card never sees anything it serves slowly. My 27B test is that gate working — one node held the model, one node got all twenty.

Pair only the similar machines. Two 3060s would split evenly and correctly.

Wait. Nvidia's README says a smarter scheduler, and likely a choice of policy, is planned.

The timeout nobody documented

I put a 27B model on the 3090 only, then sent twenty concurrent requests through PAIR. Routing was correct. Ten came back. The other ten failed at 120.2 to 120.3 s with `502 upstream error: net/http: timeout awaiting response headers`.

Ollama sends no response headers until it starts generating; at about 11 s per 200-token 27B request, anything past tenth in the queue waits more than 120 s for headers — and the proxy gives up at 120 s. The same twenty requests sent straight to Ollama all completed — in 226 s.

Eight requests through PAIR, so the deepest queue position starts at 75 s, all completed with zero failures. It is queue depth, not the model.

The same limit cancelled a cold model load: a 9 GB model on the 3060's slow disk takes two to three minutes to load, the first request through PAIR for it got the same 502 at 120 s, Ollama abandoned the load when the connection dropped, and the retry — with the file now in page cache — took 31 s.

It is not in the README, the proxy reference, its flag table, or the troubleshooting guide, and no setting changes it. The ceiling it imposes is 120 s divided by your per-request time, per node, regardless of how many requests you send. I have an issue drafted with the repro and log lines; it is not filed as of 4 September 2026, and I will add the date here when it is.

And mine did worse

I run my own router. mycoSwarm has been dispatching work across this same pair of machines and four CPU boxes for months, and I was fairly sure it would do better than a job counter. So I fanned the same twenty requests through it, three times.

0.52x. Every one of the sixty requests went to the 3060 — 101 s against 52.6 s for the 3090 alone in the same session.

The reason is in my own scoring function, and I will state it plainly because it is embarrassing in exactly the useful way. For an inference task, a node with a GPU gets 1000 points, plus VRAM divided by 100, plus 500 if it is a "specialist" node and 200 if it is the "executive". The 3060 box is a specialist — 1000 + 123 + 500 = 1623 for the 3060 against 1000 + 246 + 200 = 1446 for the 3090. In-flight load is subtracted at 100 points per job — but only from the local node's score, never from a peer's. So the 3090, which is where the request arrives, penalises itself for every job it is already running and never learns that the 3060 has nineteen queued. The tiebreak I wrote to keep small models off the big card sent every request to the slow one.

Two routers — wrong in the same direction. PAIR ignores speed. Mine ignores remote load. Either one, fed a 3090 and a 3060, produces a cluster that is no better than the 3090 by itself, and one of them produces a cluster that is half as good. The fix on my side is a task on the list, not a promise: subtract in-flight load for peers too, and weight by measured service time rather than a tier label.

What PAIR is

The repo is public under Apache 2.0 and in beta. You point an app at a local address that looks like Ollama or LM Studio, and PAIR picks the machine that serves each request. In Nvidia's own words it does not merge GPUs or pool VRAM, shard a model, or split one request across machines.

Their launch demo ran a five-subagent Hermes Desktop workload on Qwen 3.6 35B-A3B: 18 minutes on one RTX Spark laptop, 8 minutes 48 seconds on that laptop plus a DGX Spark plus an RTX 5090. That is a 2.05x from a mixed cluster measured against its weakest member alone.

Install and pairing, compressed

The Linux release is a `.deb` that wants sudo and GTK, plus a separate 85 MB archive of thirteen static Go binaries for machines with no desktop. I used the archive on both nodes and never needed root. It adopted the Ollama already on each machine rather than installing its own; with Ollama on 11434, the proxy went to 11435 with a warning. Cluster identity survived every restart.

Both machines have wired 192.168.50.x and WiFi 192.168.1.x links; PAIR picked WiFi on both, and no flag, environment variable, or setting in v0.1.1 changes it.

Without sudo, two confounds. The 3060's old Ollama 0.17.5 owned port 11434, so PAIR adopted it and could not be pointed elsewhere. Stopping that service and putting 0.30.0 on the same port took the 3060 from 5.0 s to 4.4 s per request and PAIR from 1.00x to 1.07x. The WiFi routing is still open.

Limits

Two nodes, two GPUs. One model per test: a 9.7B for the routing runs, a 27B for the timeout. Ollama on both ends, not llama.cpp, at its default single slot. PAIR v0.1.1 — a beta build a week old when I tested it. One day of measurement; every headline figure is three reps and the split never varied. Nvidia's own line is that PAIR is "a better fit for similar machines than a highly mixed cluster." I do not own two similar machines.

Method and full results

Workload. Twenty distinct prompts, each asking for about 250 words on a different local-AI topic, `max_tokens` 200, temperature 0, sent concurrently over the OpenAI-style `/v1/chat/`

`completions` route with a thread per request. Every request returned exactly 200 completion tokens. Direct runs went to the 3090's Ollama on 11434; PAIR runs to its proxy on 11435 on the same machine. Reps alternated direct, PAIR, direct, PAIR, direct, PAIR, with both models warmed and a 30-minute keep-alive first.

Model. `qwen3.5:9b` Q4_K_M, same digest on both nodes. Qwen 3.6 has no 9B, on the Ollama library or anywhere else I could find, so the 3.5 stood in. Both Ollamas run `OLLAMA_NUM_PARALLEL=1`, so "concurrent" means queued at the engine.

Attribution. PAIR's proxy logs one `proxy request complete` line per request at debug level, with node id, target address, status, and duration. Every split figure below comes from those lines, cross-checked against PAIR's persisted `workloads-history.json`. Service time per node is the last completion on that node divided by its request count.

Machines. Miu: RTX 3090 24 GB, i7-8086K, Ubuntu 24.04, Ollama 0.30.0. Rushuna: RTX 3060 12 GB, i7-7700, Ubuntu 24.04, Ollama 0.17.5 in phases 1 and 2, 0.30.0 in phase 3.

Phase 3, Ollama 0.30.0 on both nodes (headline)

Rep	Direct, 3090 only	PAIR	Split	3090 service	3060 service
1	48.05 s	44.46 s	10 / 10	2.36 s	4.42 s
2	46.66 s	43.95 s	10 / 10	2.32 s	4.36 s
3	46.56 s	43.89 s	10 / 10	2.31 s	4.36 s
Mean, SD	47.09 s, 0.83	44.10 s, 0.31			

Ratio 1.07x.

Phase 1, Ollama 0.17.5 on the 3060

Rep	Direct	PAIR	Split	3090 service	3060 service
1	50.32 s	50.27 s	10 / 10	2.45 s	5.00 s
2	48.88 s	50.20 s	10 / 10		
3	48.18 s	50.22 s	10 / 10		
Mean, SD	49.12 s, 1.09	50.23 s, 0.03			

Ratio 0.98x. A phase 2 rerun the same day gave 1.05x with a cold first direct rep and 1.00x without it.

mycoSwarm, same twenty requests

Rep	Wall	Split	3060 service
1	101.29 s	0 / 20	5.07 s
2	100.93 s	0 / 20	
3	102.00 s	0 / 20	
Mean, SD	101.41 s, 0.55		

Ratio 0.52x against the phase 2 direct mean. mycoSwarm's worker on the 3060 was still on Ollama 0.17.5 for these runs.

27B on the 3090 only, through PAIR

Requests	Direct	PAIR	Succeeded	Failed
20	226.5 s, 20/20	120.3 s	10	10, all 502 at 120.2 to 120.3 s
8	86.5 s, 8/8	85.6 s	8	0

GPU utilization on the 3090 fell to zero within a second of the 502s, so the ten orphaned requests were cancelled rather than run to waste.

Raw results, attribution logs, GPU samples, the load generator, and the issue draft are in the benchmarks record.
