Nvidia's router dealt the cards evenly. That was the whole problem.

September 8, 2026 · by Mark Bartlett

Download this post as PDF

InsiderLLM Weekly issue 20 – September 8, 2026

A day late: Monday went to the two bench runs at the bottom of this issue.

Nvidia shipped a router on Thursday for the exact problem on my desk, and by lunchtime it was splitting my requests straight down the middle between the 3090 and the 3060, every run, as if the two cards were the same card.

Quick Hits

- **PAIR's proxy drops any request still waiting for its first byte at 120 seconds, and there is no setting for it.** Twenty 27B requests through the proxy: half came back, half got a 502. Sent straight to Ollama, all twenty finished. 📖 The full PAIR bench.
- **Both bench boxes have a wired link and a WiFi link. PAIR picked WiFi on both,** and nothing in v0.1.1 lets you choose. 📖 The full PAIR bench.
- **Training loss did not predict the adapter's score.** The seed with the lowest final loss scored 19 of 47, and the two top scorers finished with the highest losses in the clean batch. 📖 The ten seeds.
- **Thirty-four of the 47 test items score the same with or without an adapter.** The whole ten-seed result lives in the other thirteen, and six of those go whichever way the seed sends them. 📖 The ten seeds.

The Router Counts Jobs, and Nothing Else

PAIR is Nvidia's Personal AI Router: free, Apache 2.0, announced Thursday; the v0.1.1 build dates from 28 August. You point an app at an address that looks like Ollama, and PAIR picks which machine on your network serves each request. It does not pool VRAM or shard a model. It routes. Nvidia's launch demo got 2.05x from a laptop, a DGX Spark and a 5090 together, measured against the laptop alone.

I paired it across my two bench machines, an RTX 3090 and an RTX 3060, in 18 seconds, with no root, and it adopted the Ollama already on each box. Then I sent twenty concurrent requests through it, three times, and it split them **10 / 10** on every run. Nine runs across three sessions and the split never varied.

That is the whole result, because the 3060 takes **4.4 s** per request and the 3090 takes **2.35 s**. Ten on the 3060 is 44 seconds, and that is the wall clock, because the 3090 was done with its ten after 23 seconds and spent the rest waiting. Give all twenty to the 3090 and it takes 47. A second machine, then, bought me **1.07x**. With the 3060 still on its old Ollama build it was 5.0 s per request, and PAIR came in 2 percent behind the 3090 running alone.

The rule falls straight out of the arithmetic: **split evenly across N nodes, the slowest one has to be under N times slower than the fastest, or you were better off with the fastest machine by itself.** A 3060 beside a 3090 sits right at that line. A load-aware split on the same numbers, thirteen requests to the fast card and seven to the slow one, would have been 1.40x, which is where I had pre-registered my expectation. Nvidia already knows. Its known-issues page says as much under the heading “Scheduling Only Counts Jobs”: on a mixed cluster, a slow node gets work about as often as a fast one.

Then I ran the same twenty requests through mycoSwarm, the router I wrote, which has been dispatching across this same pair of machines for months. **0.52x.** Every one of sixty requests went to the 3060. My scoring function gives the 3060 box 500 points for being a “specialist” and subtracts in-flight load only from the local node, never from a peer, so the 3090 penalises itself for its own queue and never learns the 3060 has nineteen waiting. A tiebreak I added to keep small models off the big card ended up sending everything to the small one.

Both routers fail the same way from opposite ends. PAIR never asks how fast a node is. Mine never asks how busy the other node is. Hand either a 3090 and a 3060 and you get a cluster that cannot beat the 3090 alone, and in my case one that does half as well.

📖 **The full method, the 502 repro, and the scoring function that did it: here.**

Ten Seeds, Same Side Every Time

Last week I had two LoRA adapters trained from one recipe and wouldn’t print the spread. Now there are ten.

Same 24 training examples, same configuration, same 27B base in 4-bit, changing only the seed. Scored against the frozen 47-item split from the prompt experiment. The ten adapters land between **15 and 21 of 47**, a **12.77-point** spread from worst to best, and the standard deviation

across them is **3.62 points**. The prompt compiler, run ten times on the same task on 30 August, spread 4.65. Narrower by a fifth, which buys nothing.

With no adapter at all, the base gets 21 of 47. Two seeds matched that. Eight came in under it. **Not one came in over.** The mean effect is minus 3.62 points, the same number as the spread, which is a coincidence I double-checked. All ten adapters memorised their 24 examples. None of them picked up the task, and on average they cost the base a little of what it already knew.

So the coin is real, and it is double-headed. What that does to the substrate question from last issue is settle it as a cost decision. When a compiled prompt and a trained adapter each give you one draw from distributions of roughly equal width, there is no reliable option, only a cheaper one. The prompt is 24 cents to compile and 1,383.9 extra tokens on every call for the life of the deployment. The adapter is sixteen and a half minutes of 3090 time per seed and nothing after that. Pick either, and plan on paying for several runs, since a single one tells you almost nothing.

The protocol I'm keeping: at least three seeds, score each of them against the bare base model along the path it trained on, report the spread and not the best seed. And never pick the seed by its loss curve.

This is the third piece in a series. The first compiled a prompt skill twenty times and found the headline gain was the top of ten draws. The second measured the prompt tax and found nobody had checked whether the weights alternative reproduces. This one checks.

📖 **Part one, the prompt compiler: here.**

📖 **Part two, the prompt tax and the LoRA substrate: here.**

📖 **Part three, the ten seeds: here.**

Running Now, Not Yet Counted

Qwen3.8-Flash-Next is on the 3090 as of Monday evening, stock llama.cpp, every routed expert pushed to system RAM and only the dense layers and shared experts on the card. It loads, it answers, and the runs are logged, and I am not printing a single figure from them until the 3060 box has had its turn, which is this week. The question is not how fast the 3090 runs it. The question is what happens to the gap between the two cards when the weights that matter live in system memory rather than VRAM, and whether the budget box, with half the RAM, runs it at all. Both rigs, same file, same build, same harness, then numbers.

Housekeeping. The benchmark dataset's llama.cpp pin moves from b10088 to v0.4.0 as of today. The gate was a back-to-back re-bench of the two canonical 3090 rows, Ornith 1.5-35B and Qwen 3.6-35B, on both builds in one session, three reps each, builds alternating. Pooled, no cell moved more than 2.4 percent. One wobbly rep in each block crossed the 3 percent line, in opposite directions, and they cancelled, which is what "no build effect at the 3 percent line" means in practice. Every published row keeps its b10088 tag; new rows carry v0.4.0. The file is still at **118 rows**. CC BY 4.0, attribution the only ask.

📖 **The dataset, and the raw JSON: here.**

That's the week. If you are about to buy a second machine to put behind a router, check how much slower it is than your first one before you check anything else. If it is more than twice as slow, the router you are looking at will make things worse, and I have written one that manages to do worse than that.

New here, reading this on the web? Subscribe and the next one lands in your inbox.

— Mark, InsiderLLM

Run PAIR on machines that aren't a 3090 and a 3060? Trained an adapter more than once and compared? Reply, or hit me at hello@insiderllm.com. I read everything.
