Multi-GPU Local AI: Run Models Across Multiple GPUs

February 6, 2026 · by Mark Bartlett

Download this guide as PDF

Quick Answer: Two GPUs don't give you twice the speed — they give you twice the VRAM. That's the point. A 70B model that can't fit on one 24GB card runs at 16-21 tok/s across dual RTX 3090s. Without multi-GPU, your only option is CPU offloading at ~1 tok/s. The three methods are tensor parallelism (splits each layer across GPUs, all compute simultaneously), pipeline parallelism (assigns entire layers to different GPUs sequentially), and managed stacks like Razer AIKit. llama.cpp and Ollama handle multi-GPU automatically. vLLM gives you the most control for production serving. For consumer hardware on PCIe, pipeline parallelism has less overhead than tensor parallelism — save tensor parallelism for NVLink setups.

📚 **More on this topic:** Razer AIKit Guide · GPU Buying Guide · VRAM Requirements · llama.cpp vs Ollama vs vLLM

You want to run a 70B model locally. Your RTX 3090 has 24GB of VRAM. The model needs 45GB at Q4 quantization. No amount of clever quantization will squeeze it onto one card.

The solution: split the model across two GPUs. Two 3090s give you 48GB of usable VRAM — enough for 70B models at Q4, or 32B models at near-lossless Q8 quality. But multi-GPU isn’t free performance. There’s communication overhead, PCIe bandwidth limitations, and configuration that varies by tool.

This guide covers every method for splitting LLMs across multiple GPUs on consumer hardware — how each approach works, what performance to actually expect, practical setups with real hardware, and the pitfalls that catch most people.

Why Multi-GPU (and When to Skip It)

Multi-GPU solves one problem: **running models that don’t fit on a single card.** That’s it.

If your model fits on one GPU, adding a second one makes it slower, not faster. Benchmarks with an 8B model on RTX 3090s:

GPUs	tok/s	vs Single GPU
1	111.7	baseline
2	108.1	-3.3%
4	104.9	-6.1%
6	101.1	-9.6%

Every GPU you add introduces communication overhead. For a model that already fits, that overhead is pure loss.

Use multi-GPU when:

- The model you need doesn't fit on your largest single GPU
- You need to serve many concurrent users (batch throughput scales well)
- You want higher quantization quality (Q8 instead of Q4) on large models

Don't use multi-GPU when:

- Your model fits on one card — just use Ollama
- You're trying to make a small model faster — buy a faster single GPU instead
- You have budget for one better GPU instead of two worse ones

The Two Approaches: Tensor vs Pipeline Parallelism

Tensor Parallelism (TP)

Tensor parallelism splits individual weight matrices within each layer across GPUs. All GPUs compute their portion of the same layer simultaneously, then combine results.

How it works: The attention and feed-forward layers contain large weight matrices. TP slices these matrices so GPU 0 computes the left half and GPU 1 computes the right half. After each layer, an AllReduce operation combines the results. This happens twice per transformer layer — once after attention, once after the feed-forward network.

Pros:

- All GPUs work simultaneously — better utilization
- Lower latency for single requests (no pipeline bubbles)

Cons:

- Two AllReduce operations per layer — heavy on inter-GPU bandwidth
- Requires high-speed interconnect (NVLink) to shine
- On PCIe, the communication overhead can negate the benefit

Pipeline Parallelism (PP)

Pipeline parallelism assigns entire layers to different GPUs in sequence. GPU 0 processes layers 0-19, sends the output to GPU 1, which processes layers 20-39.

How it works: Like an assembly line. Data flows through GPU 0 first, then GPU 1. Only one activation tensor transfers between GPUs per forward pass (not two AllReduces per layer like TP).

Pros:

- Much less inter-GPU communication
- Works well on PCIe bandwidth
- Supports unequal GPU sizes (assign more layers to the bigger card)

Cons:

- “Pipeline bubbles” — GPUs sit idle waiting for the previous stage
- Only one GPU active at a time for single requests
- Higher latency than TP for individual requests

Which to Use

Your Setup	Recommended Approach
2 GPUs with NVLink	Tensor parallelism
2 GPUs on PCIe (consumer boards)	Pipeline parallelism
Mixed GPU sizes (e.g., 3090 + 3060)	Pipeline parallelism
Multi-machine cluster	Pipeline between machines, tensor within machines
High-throughput batch serving	Tensor parallelism (amortizes communication)

vLLM’s own documentation states it directly: “if GPUs do not have NVLink interconnect, leverage pipeline parallelism instead of tensor parallelism for higher throughput and lower communication overhead.”

How Each Tool Handles Multi-GPU

llama.cpp

llama.cpp gives you the most fine-grained control over GPU splitting.

Layer split (pipeline parallelism) — default:

```
llama-server --model llama-70b-q4.gguf \
  -ngl 999 \
  --split-mode layer
```

Row split (tensor parallelism):

```
llama-server --model llama-70b-q4.gguf \
  -ngl 999 \
  --split-mode row
```

Custom split ratios for unequal GPUs:

```
# 3090 (24GB) + 3060 (12GB): assign 2/3 to GPU 0, 1/3 to GPU 1
llama-server --model llama-70b-q4.gguf \
  -ngl 999 \
  --tensor-split 24,12
```

Without `--tensor-split`, llama.cpp automatically distributes proportionally to free VRAM on each GPU.

Performance (2x RTX 4090, LLaMA 3 70B Q3_K, prompt processing):

Split Mode	tok/s
Layer (default)	1,287
Row	582
Tensor (new, experimental)	1,072

Layer split still wins for most consumer PCIe setups. The new `--split-mode tensor` (PR #19378) is experimental and improving, but layer split remains the default for good reason.

Gotchas:

- Row split requires CUDA. It does not work with Vulkan.
- The `--main-gpu 0` flag sets which GPU handles the scratch buffer and KV cache in row mode.
- Always use `-ngl 999` to offload all layers to GPUs. Partial offloading with multi-GPU creates performance cliffs.

Ollama

Ollama handles multi-GPU automatically since v0.11.5. No configuration needed — it detects your GPUs and distributes layers.

```
# Just run normally. Ollama splits automatically.
ollama run llama3.1:70b
```

How Ollama distributes:

1. Sorts GPUs by available VRAM (largest first)
2. Assigns layers starting from the output layer
3. Packs layers onto the fewest GPUs that fit

To force spreading across all GPUs:

```
OLLAMA_SCHED_SPREAD=true ollama run llama3.1:70b
```

To restrict to specific GPUs:

```
CUDA_VISIBLE_DEVICES=0,1 ollama serve
```

Limitations: Ollama only does pipeline parallelism (layer splitting). No tensor parallelism. You can’t control which layers go where. For a model that fits on one GPU, Ollama still spreads across multiple GPUs by default — there's no way to prevent this without `CUDA_VISIBLE_DEVICES`.

vLLM

vLLM provides both parallelism modes and is the best choice for production multi-GPU serving.

Tensor parallelism:

```
vllm serve meta-llama/Llama-3.3-70B-Instruct \
  --tensor-parallel-size 2
```

Pipeline parallelism:

```
vllm serve meta-llama/Llama-3.3-70B-Instruct \
  --pipeline-parallel-size 2
```

Combined (multi-node):

```
# 2 nodes, 4 GPUs each: tensor within node, pipeline across nodes
vllm serve meta-llama/Llama-3.3-70B-Instruct \
  --tensor-parallel-size 4 \
  --pipeline-parallel-size 2
```

Important: vLLM tensor parallelism requires all GPUs to have the **same VRAM**. Mixed GPU sizes only work with pipeline parallelism.

vLLM with dual RTX 3090s (tensor parallelism) runs Llama 3 70B Q4 at ~21 tok/s for single requests. For batch serving with 50 concurrent users, 8 GPUs achieve ~800 tok/s total throughput — this is where multi-GPU shines.

For a full comparison of these tools, see our llama.cpp vs Ollama vs vLLM guide.

Razer AIKit

Razer AIKit wraps vLLM, Ray, LlamaFactory, and Grafana into a Docker stack with a CLI that auto-configures multi-GPU:

```
# AIKit auto-detects your GPUs and suggests settings
rzr-aikit gpu-select meta-llama/Llama-3.1-70B-Instruct

# Run with tensor parallelism
rzr-aikit model run meta-llama/Llama-3.1-70B-Instruct \
  --tensor-parallel-size 2
```

AIKit also handles multi-machine clusters via Ray, adds Grafana monitoring for GPU utilization and token throughput, and includes LlamaFactory for fine-tuning. It's the turnkey option if you want the full stack without wiring it up yourself.

ExLlamaV2

ExLlamaV2 added tensor parallelism in v0.3.2. It uses EXL2 quantization (often more efficient than GGUF at comparable sizes):

```
python exllamav2/server.py \
  --model /path/to/model \
  --gpu_split auto
```

For specific VRAM allocations: `--gs 24,12` assigns 24GB to GPU 0 and 12GB to GPU 1. Supports mixed GPU sizes natively.

What You Can Run on Multi-GPU Setups

2x 24GB (Dual RTX 3090 / 4090) — 48GB Total

This is the most common and cost-effective multi-GPU setup.

Model	Quantization	VRAM Needed	Fits?
Llama 3.3 70B	Q3_K_M	37.5 GB	Yes
Llama 3.3 70B	Q4_K_M	45.6 GB	Yes (tight)
Qwen 2.5 72B	Q3_K_M	40.9 GB	Yes
Qwen 2.5 72B	Q4_K_M	50.5 GB	No

Model	Quantization	VRAM Needed	Fits?
Any 32B model	Q8_0	~32 GB	Yes
Any 32B model	FP16	~64 GB	No
DeepSeek R1 Distill 70B	Q4_K_M	~45 GB	Yes (tight)

The sweet spot: **70B models at Q3-Q4 quantization.** This is the class of model that justifies dual 24GB cards. Anything smaller runs fine on a single card.

Other Configurations

Setup	Total VRAM	Best Model Class
2x 8GB (16GB)	16 GB	14B at Q4 (Qwen3 14B, Phi-4)
2x 12GB (24GB)	24 GB	27-32B at Q3-Q4 (Gemma 3 27B, Qwen3 32B)
2x 16GB (32GB)	32 GB	32B at Q5-Q6 or 27B at Q8
2x 24GB (48GB)	48 GB	70B at Q3-Q4
4x 24GB (96GB)	96 GB	70B at Q8, or 123B at Q4

4x 24GB opens up 70B models at near-lossless Q8 quality, or 100B+ models at Q4. Beyond that, you're into datacenter territory — a 405B model needs 200GB+ even at Q4.

For detailed VRAM requirements per model, see our VRAM requirements guide.

→ Use our Planning Tool to check exact VRAM for your setup.

PCIe Bandwidth: The Hidden Bottleneck

Multi-GPU performance on consumer hardware is limited by how fast GPUs can communicate. Consumer GPUs use PCIe, not NVLink.

Interconnect	Bandwidth (per direction)
PCIe 3.0 x16	16 GB/s
PCIe 4.0 x16	32 GB/s
PCIe 4.0 x8 (dual GPU)	16 GB/s

Interconnect	Bandwidth (per direction)
PCIe 5.0 x16	64 GB/s
NVLink 3 (RTX 3090)	56.25 GB/s
NVLink 4 (H100)	450 GB/s

The Lane Splitting Problem

When you install two GPUs on a consumer motherboard, the CPU's 16 PCIe graphics lanes typically split to **x8 per GPU**. This halves the available bandwidth per card compared to single-GPU operation.

Some motherboards split as **x16/x4** instead — giving the second GPU only 4 GB/s. This creates a **35-40% performance penalty** for tensor parallel workloads on the second card. Check your motherboard manual before buying a second GPU.

Workstation boards (AMD Threadripper, Intel Xeon W) provide 64+ PCIe lanes and maintain full x16 per slot even with multiple GPUs. This costs more but eliminates the bandwidth bottleneck.

NVLink vs PCIe

The RTX 3090 is the last consumer NVIDIA card with NVLink support. With an NVLink bridge (~$80-120), dual 3090s get 112.5 GB/s bidirectional — roughly 3.5x the bandwidth of PCIe 4.0 x8. Real-world tests show **40-60% faster** multi-GPU inference with NVLink vs PCIe.

The RTX 4090 and 5090 removed NVLink. If you want NVLink on consumer hardware, the 3090 is your only option.

Does PCIe Generation Matter?

Not much for inference. Benchmarks show only **2-4% improvement** moving from PCIe 4.0 to 5.0 for LLM inference. The bottleneck is GPU memory bandwidth (936 GB/s on a 3090), not PCIe transfer speed. Don't buy a new motherboard for PCIe 5.0 if multi-GPU AI is your goal.

Performance: What to Actually Expect

The Scaling Reality

For models that require multi-GPU:

Setup	Model	tok/s	Notes
1x RTX 3090 + CPU offload	70B Q4	~1	Unusably slow
2x RTX 3090 (layer split)	70B Q4	~16-18	Usable for chat
2x RTX 3090 (vLLM TP)	70B Q4	~21	Best single-request speed
2x RTX 4090 (layer split)	70B Q4	~18-22	Faster compute per GPU
8x GPU (vLLM TP, batched)	70B BF16, 50 users	~800 total	Batch serving scales well

The jump from 1 to 21 tok/s is the entire value proposition. You're not getting double speed — you're going from "impossible" to "usable."

Where Overhead Comes From

1. **AllReduce communication** (tensor parallelism): Two synchronization points per transformer layer. Data transferred scales with hidden dimension.
2. **Pipeline bubbles** (pipeline parallelism): GPUs idle while waiting for the previous stage.
3. **PCIe lane splitting**: x8 per GPU instead of x16 halves available bandwidth.
4. **Synchronization barriers**: GPUs wait for each other at every communication point.
5. **Driver/library overhead**: NCCL and CUDA coordination has fixed per-transfer costs.

When Adding More GPUs Stops Helping

- **2 GPUs:** Best cost-benefit. ~5-15% overhead for pipeline, more for tensor on PCIe.
- **4 GPUs:** Still worthwhile. Enables 70B at Q8 or 100B+ at Q4.
- **8+ GPUs on PCIe:** Diminishing returns for single-request inference. Communication starts dominating. Better suited for batch throughput where you're serving many users.

Batch serving is where multi-GPU pays off most. More GPUs = more KV cache capacity = more concurrent requests. A single request doesn't benefit much beyond 2-4 GPUs, but 50 concurrent requests scale nearly linearly to 8 GPUs.

Practical Setup: Dual RTX 3090

The dual RTX 3090 is the most popular consumer multi-GPU setup for local AI. Here's what it takes.

Cost (February 2026)

Component	Cost
2x Used RTX 3090	$1,600-1,700 (~$800-850 each)
NVLink bridge (optional)	$80-120
1,200W+ PSU	$150-200
GPU total	**$1,680-$1,820**

For buying advice, see our used RTX 3090 guide and GPU buying guide.

Power

Each RTX 3090 draws 350W TDP and can peak at 400-500W under sustained inference. Two cards plus the rest of your system:

- GPUs under load: ~700-900W
- CPU + rest: ~150-250W
- **Total system peak: 1,000-1,150W**
- **Minimum PSU: 1,200W.** A 1,600W PSU gives you comfortable headroom.

Critical: Use two separate PCIe power cables per GPU. Don't use a single cable with dual 8-pin pigtails — the cable can overheat and melt at 350W+ sustained load. LLM inference keeps GPUs at full load continuously, unlike gaming which has intermittent bursts.

Cooling

Two 3-slot cards in a standard ATX case is tight. Maintain at least one slot of spacing between cards — testing shows 15 degrees C lower temperatures vs cards touching. Blower-style coolers exhaust heat directly out of the case, which is better for multi-GPU than open-air coolers that dump heat into the chassis.

Plan for 240mm of intake airflow per GPU at minimum. Sustained inference at 350W per card 24/7 is harder on cooling than gaming.

Motherboard

Consumer AMD and Intel boards provide 16 PCIe lanes for graphics. With two GPUs, most boards split to x8/x8 — check your manual. Some boards split x16/x4, which drops second-GPU performance by 35-40%.

For optimal multi-GPU, workstation boards (AMD Threadripper WRX90, Intel W790) provide 64+ PCIe lanes with full x16 per slot. This adds $500-1,000 to your build but eliminates the bandwidth bottleneck.

Mixed GPU Setups

Can you run a 3090 (24GB) and a 3060 (12GB) together? Yes — with caveats.

What Works

llama.cpp: Full support. `--tensor-split 24,12` assigns work proportionally to VRAM.

Ollama: Automatic. Distributes layers based on available VRAM per GPU.

ExLlamaV2: Full support. `--gs 24,12` for specific allocation.

What Doesn't Work

vLLM tensor parallelism: Requires identical VRAM across GPUs. Mixed sizes fail to load. Workaround: use pipeline parallelism (`--pipeline-parallel-size 2, --tensor-parallel-size 1`).

The Performance Trade-off

Mixed GPUs give you more total VRAM, but performance is pulled toward the slower card. A 3090 (936 GB/s memory bandwidth) paired with a 3060 (360 GB/s) means:

- In pipeline parallelism: the 3060 becomes a bottleneck for every forward pass
- In tensor parallelism: the 3090 waits for the 3060 at every AllReduce

A 3090 + 3060 gives you 36GB total VRAM and runs 70B models, but slower than dual 3090s with 48GB. If you already have both cards, use them. If you're buying, two identical GPUs are always better.

Common Pitfalls

PCIe Lane Splitting

Most people don’t check their motherboard’s PCIe lane configuration before buying a second GPU. If your board splits x16/x4 (common on budget boards), the second GPU gets 4 GB/s bandwidth instead of 16 GB/s. Verify before buying.

Power Delivery

Two RTX 3090s can draw over 900W at peak. A 750W PSU will shut down under load. LLM inference is sustained full-load, not intermittent like gaming. Budget for a 1,200W+ PSU and dedicated power cables.

Thermal Throttling

GPUs touching each other with no airflow gap will thermal throttle within minutes under sustained inference. Leave at least one slot of spacing. Consider water cooling for truly sustained 24/7 workloads.

“I Added a GPU and It’s Slower”

If your model fits on one GPU, adding a second GPU adds overhead without benefit. This is the most common mistake. Multi-GPU only helps when the model needs more VRAM than one card provides.

Vulkan Multi-GPU

Don’t use Vulkan for multi-GPU in llama.cpp. Row split doesn’t work, there are crash reports on recent builds, and it can only access ~4GB of an 8GB GPU. Use CUDA.

Mixed NVIDIA + AMD

No tool supports splitting a model across an NVIDIA and an AMD GPU simultaneously. Pick one brand for your multi-GPU setup.

The Bottom Line

Multi-GPU local AI has one real use case: running models that don't fit on a single card. The dual RTX 3090 at 48GB total VRAM unlocks 70B models — a class of model that's genuinely more capable than anything you can run on 24GB alone.

The practical summary:

Scenario	Best Approach
"I want 70B on dual 3090s"	llama.cpp layer split or vLLM pipeline parallelism
"I have dual 3090s with NVLink"	vLLM tensor parallelism
"I want the easiest setup"	Ollama (auto-splits, no config)
"I want monitoring and fine-tuning too"	Razer AIKit
"I have mixed GPUs (3090 + 3060)"	llama.cpp with `--tensor-split 24,12`
"I'm serving many users"	vLLM with tensor parallelism

Don't buy a second GPU to make a small model faster. Buy a second GPU to run a bigger model. That's the only equation that works.

Related Guides

- Razer AIKit Guide
- GPU Buying Guide for Local AI
- VRAM Requirements for Local LLMs
- llama.cpp vs Ollama vs vLLM
- What Can You Run on 24GB VRAM
- Used RTX 3090 Buying Guide
- Fine-Tuning on Consumer Hardware
- Quantization Explained

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/multi-gpu-local-ai/