Razer AIKit Guide: Multi-GPU Local AI on Your Desktop

February 6, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Razer AIKit is an open-source Docker stack (Apache 2.0) that bundles vLLM, LlamaFactory, Ray, Grafana, and Open WebUI into one container. It handles multi-GPU tensor parallelism, distributed inference across machines, LoRA fine-tuning, and real-time monitoring. The CLI auto-detects your GPUs and suggests optimal configurations. It's overkill for single-GPU hobbyists (use Ollama instead), but if you have 2+ GPUs and want production-grade serving with fine-tuning and monitoring in one stack, AIKit saves you from wiring up five different tools yourself. Runs on Windows 11 via WSL2 or Ubuntu 22.04/24.04. Minimum 8GB VRAM, but you really want 24GB+ across your GPUs to justify the complexity.

📚 **More on this topic:** GPU Buying Guide · VRAM Requirements · llama.cpp vs Ollama vs vLLM · Fine-Tuning on Consumer Hardware · Planning Tool

Running one model on one GPU is a solved problem. Ollama handles that in a single command. But the moment you want to split a 70B model across two GPUs, fine-tune it on your own data, and monitor token throughput in real time — you're stitching together vLLM, Ray, LlamaFactory, Prometheus, and Grafana by hand. It works, but it takes a full weekend.

Razer — yes, the gaming peripherals company — open-sourced AIKit in late 2025. It's a Docker-based stack that bundles all of those tools into one container with a CLI that auto-detects your GPUs and configures everything. Apache 2.0 license, supports 280K+ HuggingFace models, and runs on Windows via WSL2 or Ubuntu.

This guide covers what AIKit actually includes, who should use it, how to set it up, what you can do with it, and when simpler tools are the better choice.

What Razer AIKit Is

AIKit is a Docker container that packages a complete AI development environment:

Component	What It Does	Version (v0.2.0)
vLLM	High-throughput inference engine with OpenAI-compatible API	0.14.1

Component	What It Does	Version (v0.2.0)
LlamaFactory	Fine-tuning framework (LoRA, QLoRA, full)	0.9.4
Ray	Distributed computing for multi-GPU/multi-node clusters	2.50.1
Grafana	Real-time dashboard for GPU utilization, throughput, latency	Pre-configured
Prometheus	Metrics collection (5-second scrape interval)	Pre-configured
Open WebUI	ChatGPT-like web interface for chatting with models	Pre-configured
Jupyter Lab	Interactive notebooks for inference, fine-tuning, and experiments	Pre-configured
PyTorch	Deep learning framework	2.8.0 + CUDA 12.9

The `rzr-aikit` CLI wraps everything with commands like `model run`, `model generate`, `cluster run`, and `cluster join`. It auto-discovers your GPUs and suggests optimal tensor parallelism settings for whatever model you want to run.

License: Apache 2.0 — fully open source, commercial use allowed.

GitHub: razerofficial/aikit

Who This Is For (and Who Should Skip It)

Use AIKit if you:

- Have 2+ NVIDIA GPUs and want to run models that don't fit on one card
- Want fine-tuning and inference in the same environment
- Need production-grade serving with monitoring and an OpenAI-compatible API
- Plan to cluster multiple machines for distributed inference
- Want a managed Docker stack instead of wiring up tools manually

Skip AIKit if you:

- Have one GPU and just want to chat with models — Ollama is simpler by an order of magnitude
- Run GGUF-quantized models and want maximum single-GPU speed — llama.cpp is lighter
- Don't want Docker overhead
- Have an AMD GPU — AIKit requires NVIDIA (compute capability 7.0+)

AIKit is a power-user tool. It adds real value when you need multi-GPU distribution, fine-tuning, and monitoring in one stack. For single-GPU inference, it's unnecessary complexity.

Requirements

Hardware

- **GPU:** NVIDIA with compute capability 7.0+ (Turing or newer)
  - **Minimum:** RTX 2060 / GTX 1650 Ti (Turing) with 8GB+ VRAM
  - **Recommended:** RTX 3080+ / RTX 4070+ with 24GB+ total VRAM across GPUs
  - Also supports datacenter cards: A100, L40S, H100
- **RAM:** 8GB minimum, 32GB+ recommended
- **Storage:** 50GB free (200GB+ recommended — models are large)
- **CPU:** Any modern x86_64

Supported GPUs

Architecture	Cards	Compute Capability
Turing	RTX 2060/2070/2080, Titan RTX, T4	7.5
Ampere	RTX 3060/3070/3080/3090, A100, A6000	8.0 / 8.6
Ada Lovelace	RTX 4060/4070/4080/4090, L40S	8.9
Blackwell	RTX 5090, RTX PRO 6000	10.x

Software

- **Windows 11** with WSL2, or **Ubuntu 22.04/24.04**
- Docker Engine
- NVIDIA GPU drivers (latest recommended)
- NVIDIA Container Toolkit
- HuggingFace account + access token (for downloading models)

Setup Walkthrough

Option A: Quick Start (Single Container)

The fastest way to try AIKit:

```
# Create cache directory for model downloads
mkdir -p $HOME/.cache/huggingface

# Run the container
docker run -it \
  --restart=unless-stopped \
  --gpus all \
  --ipc host \
  --network host \
  --mount type=bind,source=$HOME/.cache/huggingface,target=/home/Razer/.cache/huggingface \
  --env HUGGING_FACE_HUB_TOKEN=<YOUR_TOKEN> \
  razerofficial/aikit:latest
```

Once inside the container:

```
# Run your first model
rzr-aikit model run deepseek-ai/deepseek-coder-1.3b-instruct

# Generate text
rzr-aikit model generate "Write a Python function to add two numbers."
```

Option B: Full Stack with Monitoring (Docker Compose)

This gives you Grafana, Prometheus, Open WebUI, and Jupyter alongside the inference engine:

```
git clone https://github.com/razerofficial/aikit.git && cd aikit

mkdir -p $HOME/.cache/huggingface
export HUGGING_FACE_HUB_TOKEN=<YOUR_TOKEN>

docker compose -f docker_compose/docker-compose.yaml up -d --pull always
```

After the containers start, you get:

Service	URL	Notes
Open WebUI	http://localhost:1919	Chat interface (connects to vLLM)
Grafana	http://localhost:3000	Monitoring dashboard (admin/admin)
Jupyter Lab	http://localhost:8888	Interactive notebooks
vLLM API	http://localhost:8000	OpenAI-compatible endpoints
Ray Dashboard	http://localhost:8265	Cluster management
Prometheus	http://localhost:9090	Raw metrics

Windows (WSL2) Setup

Windows requires a few extra steps:

1. Install latest NVIDIA GPU drivers on Windows
2. Install WSL2: `wsl --install -d Ubuntu-24.04`
3. **Enable mirrored networking** — create `%USERPROFILE%\.wslconfig`:

```
[wsl2]
networkingMode=mirrored
```

4. **Allow firewall access:**

```
Set-NetFirewallHyperVVMSetting -Name '{40E0AC32-46A5-438A-A0B2-2B479E8F2E90}' -DefaultInboundAction Al
```

5. Inside WSL, install Docker Engine and NVIDIA Container Toolkit
6. Follow Option A or B from above

WSL2 gotcha: After starting a model, you may need to run `python -m http.server --bind 0.0.0.0 8000` to trigger the WSL2 network bridge. It will error (port occupied) — that's expected. Then `rzr-aikit model generate` works. This is a known WSL2 networking quirk.

What You Can Do With It

Distributed Inference Across GPUs

This is AIKit's primary value. Split large models across multiple GPUs with tensor parallelism:

```
# Run a 14B model across 2 GPUs
rzr-aikit model run Qwen/Qwen3-14B \
  --tensor-parallel-size 2

# Run a 70B model across 4 GPUs with Ray backend
rzr-aikit model run meta-llama/Llama-3.1-70B-Instruct \
  --tensor-parallel-size 4 \
  --distributed-executor-backend ray
```

The CLI's `gpu-select` command analyzes your hardware and the model to recommend settings:

```
# Let AIKit figure out the optimal configuration
rzr-aikit gpu-select meta-llama/Llama-3.1-70B-Instruct
```

For multi-machine clusters, start a Ray head node on one machine and join workers from others:

```
# Machine 1 (head)
rzr-aikit cluster run --ifname eth0

# Machine 2 (worker)
rzr-aikit cluster join --ifname eth0 --address 192.168.1.100:6379
```

Fine-Tuning with LlamaFactory

LoRA fine-tuning is built in. No separate install, no dependency conflicts:

```
llamafactory-cli train \
  --stage sft --do_train \
  --model_name_or_path Qwen/Qwen2.5-0.5B-Instruct \
```

```
  --template qwen \
  --finetuning_type lora \
  --lora_rank 8 --lora_alpha 16 --lora_dropout 0.05 \
  --lora_target q_proj,k_proj,v_proj,o_proj \
  --dataset alpaca_gpt4 \
  --dataset_dir ~/fine-tuning/data \
  --output_dir ~/fine-tuning/adapters/lora_alpaca_gpt4 \
  --num_train_epochs 1 \
  --per_device_train_batch_size 1 \
  --gradient_accumulation_steps 8 \
  --learning_rate 2e-4 \
  --fp16
```

Then serve your fine-tuned model with the LoRA adapter:

```
rzr-aikit model run Qwen/Qwen2.5-0.5B-Instruct \
  --enable-lora \
  --lora-modules alpaca=~/fine-tuning/adapters/lora_alpaca_gpt4
```

LlamaFactory supports LoRA, QLoRA (4/8-bit), full parameter fine-tuning, DoRA, and more. It covers 100+ model families including LLaMA, Qwen, Mistral, DeepSeek, and Phi. For more on fine-tuning methods, see our fine-tuning guide.

Monitoring with Grafana

The pre-configured Grafana dashboard (port 3000) tracks:

- **GPU utilization** — per-GPU percentage in real time
- **GPU memory usage** — used vs total VRAM per card
- **Output token throughput** — tokens generated per second
- **Time to First Token (TTFT)** — latency before generation starts
- **Inter-Token Latency (ITL)** — time between each generated token
- **Request load** — requests per second hitting the API

Prometheus scrapes metrics every 5 seconds from vLLM and Ray. This matters because multi-GPU setups can have unbalanced load — one GPU maxed out while others idle — and you need visibility to diagnose it.

OpenAI-Compatible API

vLLM exposes standard OpenAI endpoints at port 8000. Any tool that speaks the OpenAI API works:

```
from openai import OpenAI

client = OpenAI(
    base_url="http://localhost:8000/v1",
    api_key="sk-dummy"  # Required but not validated
)

response = client.chat.completions.create(
    model="Qwen/Qwen2.5-7B-Instruct",
    messages=[{"role": "user", "content": "Explain tensor parallelism in one paragraph."}]
)
print(response.choices[0].message.content)
```

Interactive Notebooks

AIKit ships 8 Jupyter notebooks covering single-GPU inference, distributed inference (head + worker), LoRA fine-tuning (local and distributed), OpenAI API integration, and semantic search. Accessible at port 8888.

AIKit vs Ollama vs Plain vLLM

	Razer AIKit	Ollama	vLLM (Direct)
Setup	Docker Compose, 5-10 min	`curl + ollama run`, 2 min	Manual Python env, 30+ min
Multi-GPU	Tensor + pipeline parallelism	No	Yes (manual config)
Multi-Node	Ray clusters	No	Yes (manual config)
Fine-Tuning	LlamaFactory built in	No	No (separate tool)
Monitoring	Grafana + Prometheus	None	Manual setup
Web UI	Open WebUI + Jupyter	None (CLI only)	None

	Razer AIKit	Ollama	vLLM (Direct)
API	OpenAI-compatible (vLLM)	OpenAI-compatible	OpenAI-compatible
Model Format	HuggingFace native	GGUF (llama.cpp)	HuggingFace native
Throughput (batched)	High (PagedAttention)	Low (single request)	High (PagedAttention)
VRAM Efficiency	Good (FP16/BF16)	Best (GGUF quantized)	Good (FP16/BF16)
Complexity	Medium-High	Low	High
Best For	Multi-GPU dev environments	Single-GPU hobbyists	Production serving

Choose Ollama if you have one GPU and want to chat with models. It's simpler, uses GGUF quantization for better VRAM efficiency, and works in 2 minutes. See our Ollama vs LM Studio comparison or llama.cpp vs Ollama vs vLLM.

Choose AIKit if you have multiple GPUs, want fine-tuning in the same environment, or need production-grade monitoring and API serving.

Choose plain vLLM if you already know vLLM and want full control without Docker wrapping your stack.

Multi-GPU VRAM Considerations

Tensor parallelism splits model weights across GPUs. Two 24GB GPUs give you roughly 48GB of usable VRAM for model weights, minus overhead for KV cache and activations.

GPU Setup	Total VRAM	Can Run (FP16)	Can Run (Q4)
1x RTX 3090	24 GB	Up to ~12B	Up to ~24B
2x RTX 3090	48 GB	Up to ~24B	Up to ~48B
2x RTX 4090	48 GB	Up to ~24B	Up to ~48B
4x RTX 4090	96 GB	Up to ~48B	Up to ~96B
1x A100 80GB	80 GB	Up to ~40B	Up to ~80B

The PCIe reality check: Consumer GPUs connect through PCIe, not NVLink. This means inter-GPU communication for tensor parallelism runs at ~64 GB/s (PCIe 4.0 x16) instead of 600-900 GB/s (NVLink). You'll see diminishing returns with more GPUs, especially for smaller models where communication overhead dominates.

Practical guidance:

- **2 GPUs over PCIe** works well. The throughput penalty is modest (~10-20%) compared to the benefit of running a larger model.
- **4 GPUs over PCIe** works but communication overhead increases. Use pipeline parallelism (`--pipeline-parallel-size`) instead of tensor parallelism when possible — it communicates less between GPUs.
- **Mixed GPU setups** (different models or different VRAM) are supported in AIKit v0.2.0 with GPU ordering and load balancing.

For GPU buying recommendations, see our GPU buying guide and VRAM requirements guide.

Known Limitations

AIKit is version 0.2.0. Keep these in mind:

Early project. 74 GitHub stars, no independent reviews or benchmarks published. The performance claims are Razer's own. The community is essentially nonexistent — no Reddit threads, no Hacker News discussions.

NVIDIA only. No AMD ROCm support. If you have AMD GPUs, this isn't for you.

Large Docker image. It contains vLLM, Ray, LlamaFactory, PyTorch, CUDA, Jupyter, Conda, and Open WebUI. The v0.2.0 update reduced image size by 24.3%, but it's still substantial.

WSL2 networking quirks. The port-triggering workaround for Windows is hacky. If you're on Windows and this matters to you, expect friction.

Low VRAM (<10GB) on Windows. You'll need to reduce GPU memory utilization to 0.8 with `--gpu-memory-utilization 0.8` to avoid OOM errors.

No GGUF support. AIKit uses HuggingFace-format models through vLLM, not GGUF. If you want GGUF quantization (which fits more model into less VRAM), use Ollama or llama.cpp instead.

The Bottom Line

Razer AIKit solves a real problem: setting up multi-GPU inference with fine-tuning, monitoring, and an API is genuinely tedious without it. The Docker stack eliminates hours of dependency wrangling, and the `gpu-select` auto-configuration is helpful for choosing tensor parallelism settings.

But it's for a specific audience. If you run one GPU and just want to chat with models, AIKit is a sledgehammer for a thumbtack — Ollama does that in two minutes. If you already have a working vLLM setup, AIKit wraps it in Docker without adding much new capability.

Use AIKit when:

- You have 2+ NVIDIA GPUs and want to run models that don't fit on one card
- You want fine-tuning and inference in one managed environment
- You need Grafana monitoring for throughput and GPU utilization
- You're setting up a multi-machine cluster with Ray

Skip AIKit when:

- You have one GPU — use Ollama
- You have AMD GPUs — not supported
- You want maximum VRAM efficiency — GGUF via llama.cpp is better
- You prefer minimal overhead — plain vLLM gives you more control

AIKit is early (v0.2.0) and unproven in the wild. But the stack it assembles is solid — vLLM, Ray, and LlamaFactory are all battle-tested individually. Razer packaged them well. If multi-GPU local AI is your use case, it's worth trying before building the same thing from scratch.

Related Guides

- GPU Buying Guide for Local AI
- VRAM Requirements for Local LLMs
- llama.cpp vs Ollama vs vLLM
- Fine-Tuning on Consumer Hardware
- Run Your First Local LLM
- Ollama vs LM Studio
- What Can You Run on 24GB VRAM

- Local AI Planning Tool — VRAM Calculator

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/razer-aikit-guide/