APPENDIX B

Profiling and Performance Tool Quick Reference

Command-by-symptom lookup · DCGM field IDs · NCCL env variables · vLLM metrics · AMD equivalents · Storage diagnostics

AI/ML Infrastructure from Silicon to Scale — Venkat Vinjam · 2026 Second Edition

---

> *"Know which tool answers which question before you touch a keyboard. The wrong tool wastes hours. The right tool answers in minutes."*

This appendix is a field guide, not a tutorial. Every section answers the question: *given a specific symptom, what do I run, what do I look at, and what does the number mean?* Detailed explanations of each tool live in **Chapter 17 (Telemetry and Observability)**; use this appendix as your production-desk quick reference.

> **HOW TO USE THIS APPENDIX**
>
> Start at **B.1 (Tool Selection Tree)** — every diagnostic question maps to a specific tool. Jump directly to that tool's section for the exact command and metric interpretation. DCGM field IDs in **B.4** are validated against DCGM 3.x on H100; verify field availability on A100 or MI300X.

B.1 Tool Selection by Diagnostic Question

The single most important skill in production debugging is tool selection. Using `ncu` to diagnose an AllReduce hang, or `nsys` to investigate a kernel micro-optimization, costs 30–60 minutes per mistake. Use this table as your first lookup.

Diagnostic Question → Correct Tool Decision Tree

QUESTION	PRIMARY TOOL	FIRST METRIC TO CHECK
"Where does time go in my training step?"	nsys	Timeline idle gaps, NCCL spans
"Why is this specific kernel slow?"	ncu	SpeedOfLight roofline position
"Is my GPU healthy across the fleet?"	dcgmi dmon	Fields 203,1003,1004,1005,1009
"Is my AllReduce overlapped with backward?"	nsys+NCCL trace	NCCL spans vs backward kernel spans
"What is my cluster AllReduce bandwidth?"	nccl-tests	busbw GB/s vs expected baseline
"Why is inference TTFT P99 spiking?"	vLLM metrics	time_to_first_token_seconds P99
"Is KV cache at risk of cascade eviction?"	vLLM alert	gpu_cache_usage_perc > 0.82 CRITICAL
"Is my GPU in thermal throttle?"	DCGM field 100	THERMAL_VIOLATION > 0 + SM clock drop
"Is my model compute- or bandwidth-bound?"	ncu Roofline	SpeedOfLight_RooflineChart section
"Is my NVLink degraded?"	DCGM field 1011	NVLINK_TX_BYTES drop from baseline
"Is my Lustre filesystem healthy?"	lfs df + fio	lfs df -h; fio sequential read MB/s
"Is my operator fused?"	nsys timeline	One kernel or many? kernel count per op
"Is data loading a bottleneck?"	nsys osrt trace	CPU idle gaps between CUDA launches
"Is my IB link healthy?"	ibstatus	State: ACTIVE; Physical: POLLING = error

Fig B-1. Tool selection decision matrix — map every diagnostic question to the single correct tool. Using the wrong tool is the most common source of wasted debugging time.

B.2 Nsight Systems (nsys) — System-Level Timeline

> **Purpose:** System-wide timeline showing where wall-clock time goes — kernel launches, memory copies, NCCL collectives, CPU activity. **Answer: "Where is the time?"**
> **Overhead:** 1.5–3× slowdown. Capture 30–60 seconds in production-equivalent conditions.

Core Commands

```
# ── STANDARD TRAINING PROFILE ──────────────────────────────────────
nsys profile \
  --trace=cuda,nvtx,nccl,osrt \       # CUDA kernels + NVTX markers + NCCL + OS runtime
  --sample=cpu \                      # CPU call-stack sampling
  --output=/tmp/profile_%p \          # Separate file per rank (%p = PID)
  --duration=30 \                     # 30-second capture window
  python train.py

# ── MINIMAL (lowest overhead) ──────────────────────────────────────
nsys profile --trace=cuda --output=/tmp/trace python script.py
```

```
# ── WITH NVTX MARKERS (requires torch.cuda.nvtx.mark() in code) ──────────────
nsys profile --trace=cuda,nvtx --output=/tmp/trace python train.py

# ── ATTACH TO RUNNING PROCESS ─────────────────────────────────────────────────
nsys profile -p $(pgrep python) --duration=30 --output=/tmp/attached

# ── OPEN IN GUI ───────────────────────────────────────────────────────────────
nsys-ui /tmp/profile_12345.nsys-rep

# ── CLI STATISTICS SUMMARY ────────────────────────────────────────────────────
nsys stats /tmp/profile_12345.nsys-rep --report gputrace
# Output: top kernels by duration (% of total GPU time)
```

What to Look for in the Timeline

Observation	Diagnosis	Action
GPU row is **dense** — no visible gaps	✅ Compute-bound or bandwidth-bound — healthy	No action needed; verify with ncu roofline
Gaps > 1 ms between kernels	❌ CPU launch overhead or data pipeline stall	Use CUDA Graphs or `torch.compile` to eliminate gaps
NCCL spans overlap backward kernels	✅ Gradient communication correctly hidden	Measure overlap fraction — target > 80%
NCCL spans are **sequential** after backward	❌ Allreduce not overlapped — 10–30% throughput loss	Enable `gradient_as_bucket_view=True` in DDP
CPU idle gaps between GPU launches	❌ Data loading bottleneck or Python GIL	Prefetch with `pin_memory=True`, increase `num_workers`
Many tiny kernels (< 50 µs each)	❌ Op-by-op execution — poor fusion	Enable `torch.compile` or manual kernel fusion

NVTX Instrumentation (Add to Training Loop)

```
import torch

# Mark iteration boundaries — nsys shows step time breakdown
for step, batch in enumerate(dataloader):
    torch.cuda.nvtx.range_push(f"step_{step}")

    torch.cuda.nvtx.range_push("data_to_device")
    inputs = batch.to("cuda")
    torch.cuda.nvtx.range_pop()

    torch.cuda.nvtx.range_push("forward")
    loss = model(inputs)
```

```
    torch.cuda.nvtx.range_pop()

    torch.cuda.nvtx.range_push("backward")
    loss.backward()
    torch.cuda.nvtx.range_pop()

    torch.cuda.nvtx.range_push("optimizer")
    optimizer.step(); optimizer.zero_grad()
    torch.cuda.nvtx.range_pop()

    torch.cuda.nvtx.range_pop()   # step_N
```

B.3 Nsight Compute (ncu) — Kernel-Level Hardware Counters

> **Purpose:** Deep hardware counter analysis for a single kernel — Tensor Core utilization, DRAM bandwidth efficiency, SM occupancy, roofline position. **Answer: "Why is THIS kernel slow?"**
> **Overhead:** 5–20× slowdown. Profile specific kernels, not entire training runs.

Core Commands

```
# ── FULL KERNEL ANALYSIS ──────────────────────────────────────────
ncu \
  --kernel-name "flash_attention_fwd" \  # Target kernel (regex match)
  --launch-skip 5 \                      # Skip 5 warmup launches
  --launch-count 3 \                     # Profile next 3 launches
  --set full \                           # All hardware counters
  --section SpeedOfLight_RooflineChart \  # Roofline analysis
  --section MemoryWorkloadAnalysis \      # Memory hierarchy breakdown
  -o /tmp/kernel_profile \               # Output file
  python infer.py

# ── QUICK ROOFLINE CHECK ──────────────────────────────────────────
ncu --kernel-name X --set roofline --launch-count 1 python script.py

# ── TENSOR CORE UTILIZATION (key metric for GEMM health) ──────────
ncu --metrics sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_elapsed \
    --kernel-name matmul python script.py

# ── OPEN RESULT IN GUI ────────────────────────────────────────────
ncu-ui /tmp/kernel_profile.ncu-rep

# ── COMPARE TWO PROFILES (regression detection) ──────────────────
ncu-ui --diff baseline.ncu-rep current.ncu-rep
# Red = regression, Green = improvement

# ── CSV OUTPUT FOR CI PIPELINES ──────────────────────────────────
```

```
ncu --kernel-name X --csv --metrics dram__bytes_read.avg,sm__throughput.avg \
    python script.py > /tmp/metrics.csv
```

Key Metrics and Interpretation

Metric	DCGM / ncu Name	Healthy Value	Problem Threshold
SM throughput	`sm__throughput.avg.pct_of_peak_sustained_elapsed`	> 70%	< 40% → likely memory-bound
Tensor Core utilization	`sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_elapsed`	> 60% (BF16 GEMM)	< 20% → wrong dtype or unaligned dims
DRAM bandwidth	`dram__bytes_read.avg`	Near HBM peak (3.35 TB/s H100)	< 50% peak → cache thrashing or small access patterns
L1 cache hit rate	`l1tex__t_sector_hit_rate.pct`	> 80%	< 50% → poor locality, consider tiling
SM occupancy	`sm__warps_active.avg.pct_of_peak_sustained_active`	> 50%	< 25% → occupancy limiter (register or smem pressure)
Arithmetic intensity	SpeedOfLight section	Above ridge point (200 FLOP/byte H100)	Below ridge point → bandwidth-bound

> **Roofline interpretation:** A kernel plotting *below and left* of the roofline ceiling is bandwidth-bound — more data reuse or tiling can help. A kernel touching the *compute ceiling* is doing as well as the hardware allows. Tensor Core underutilization with high SM activity usually means FP32 fallback or non-aligned matrix dimensions (must be multiples of 8 for BF16 Tensor Cores, multiples of 16 for INT8).

B.4 DCGM — Fleet GPU Health and the Critical Field IDs

> **Purpose:** Continuous hardware health monitoring across the entire GPU fleet. Exposes temperature, power, ECC errors, clock throttling, and compute activity at 1-second resolution.

> ⚠ **Field 203 (GPU_UTIL) Is Misleading** — DCGM field 203 reports the percentage of time *any kernel was active*, sampled at 1 ms windows. A tiny kernel running for 1 µs every 1 ms reports 100% GPU utilization. Always use **field 1003 (SM_ACTIVE)** for true compute occupancy.

Essential dcgmi dmon Commands

```
# ── PRODUCTION HEALTH MONITOR (run continuously) ──────────────────────
dcgmi dmon \
  -e 203,1002,1003,1004,1005,1009,100,155,1001 \
  -d 1000 \           # 1-second interval
  -c 3600             # 1-hour capture (drop -c for continuous)

# ── QUICK SPOT-CHECK (all GPUs on node) ──────────────────────────────
nvidia-smi dmon -s pucvmet -d 1 -c 30

# ── REMOTE FLEET MONITORING ──────────────────────────────────────────
dcgmi group -c training_group
dcgmi group -g 1 -a all
dcgmi dmon -g 1 -e 1003,1004,1005,1009 -d 1000
```

Complete DCGM Field ID Reference

Field ID	Name	Unit	Healthy Range	Alert Threshold
`100`	DCGM_FI_DEV_GPU_TEMP	°C	< 80°C	> 83°C → throttle risk
`101`	DCGM_FI_DEV_MEM_MAX_OP_TEMP	°C	< 85°C	> 90°C → HBM thermal limit
`150`	DCGM_FI_DEV_POWER_USAGE	W	Varies by workload	Near TDP limit consistently
`155`	DCGM_FI_DEV_TOTAL_ENERGY_CONSUMPTION	mJ (cumulative)	Counter	Use rate() in Prometheus
`160`	DCGM_FI_DEV_XID_ERRORS	Count	0	> 0 → investigate immediately
`190`	DCGM_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L0	Count	0	> 0 → cable/ connector issue
`203`	DCGM_FI_DEV_GPU_UTIL ⚠	%	N/A (misleading)	Use field 1003 instead
`252`	DCGM_FI_DEV_CLOCK_THROTTLE_REASONS	Bitmask	0	> 0 → decode bitmask below
`1002`	DCGM_FI_PROF_GR_ENGINE_ACTIVE	%	> 85% (training)	< 60% → overhead-bound
`1003`	DCGM_FI_PROF_SM_ACTIVE	%	> 85% (training)	< 50% → severe underutilization
`1004`	DCGM_FI_PROF_SM_OCCUPANCY	%	> 50%	< 25% → occupancy limiter
`1005`	DCGM_FI_PROF_PIPE_TENSOR_ACTIVE	%	> 60% (training BF16)	< 20% → Tensor Core unused
`1006`	DCGM_FI_PROF_PIPE_FP64_ACTIVE	%	~0 (AI workloads)	> 5% → FP64 fallback bug
`1009`	DCGM_FI_PROF_DRAM_ACTIVE	%	Workload-dependent	Constant 100% → BW bottleneck
`1011`	DCGM_FI_PROF_NVLINK_TX_BYTES	B/s	Near 450 GB/s (H100×8)	< 350 GB/s → degradation
`1012`	DCGM_FI_PROF_NVLINK_RX_BYTES	B/s	Near 450 GB/s (H100×8)	< 350 GB/s → degradation

Clock Throttle Reason Bitmask Decode

```
# Field 252 bitmask — decode to find WHY clocks dropped:
0x0000000000000001  HW_SLOWDOWN             GPU too hot — reduce workload or improve cooling
0x0000000000000002  SW_POWER_CAP            Power cap set below workload demand (check nvidia-smi -q
-d POWER)
0x0000000000000004  HW_POWER_BRAKE_SLOWDOWN Power from upstream source (PSU/switch) limited
0x0000000000000008  SYNC_BOOST              Clock sync across multiple GPUs (normal in some configs)
0x0000000000000010  SW_THERMAL_SLOWDOWN     SW thermal limit reached (85°C default)
0x0000000000000020  HW_THERMAL_SLOWDOWN     HW thermal emergency — immediately reduce workload
0x0000000000000040  DISPLAY_CLOCKS_SETTING  Display active (remove from compute node)

# Prometheus query: find all throttled GPUs
DCGM_FI_DEV_CLOCK_THROTTLE_REASONS > 0
```

XID Error Reference (Most Common)

XID	Name	Severity	Action
`48`	DBE (Double-Bit ECC Error)	CRITICAL	Immediate workload evacuation; hardware replacement required
`63`	ECC page retirement needed	HIGH	Schedule maintenance; retire affected pages
`74`	NVLINK error	HIGH	Check cables and peer GPU; disable link if persistent
`79`	GPU failure requiring reset	CRITICAL	GPU reset required; drain job from node
`80`	Correctable SBE (Single-Bit ECC)	LOW	Monitor rate; alert if > 1,000/hr
`94`	Contained ECC error, not corrected	HIGH	Drain node after workload ends
`95`	Uncontained ECC error (possible data loss)	CRITICAL	Immediate evacuation required

```
# Monitor XID errors in production:
dmesg -T | grep -i "NVRM.*Xid"
# or via DCGM:
dcgmi dmon -e 160 -d 1000 -c 60    # XID counter (field 160)
```

SM Activity Diagnostic Matrix

SM_ACTIVE (1003)	TENSOR_ACTIVE (1005)	DRAM_ACTIVE (1009)	Diagnosis
High (>85%)	High (>60%)	Low	✅ HEALTHY — compute-bound training
High	High	High	✅ HEALTHY — bandwidth-bound decode (correct)
High	Low	Low	⚠️ Tensor Cores unused: wrong dtype or misaligned dims
High	Low	High	⚠️ Memory-bound, no Tensor Core work: elementwise-dominated
Low	Low	Low	❌ Launch overhead or data loading stall: nsys timeline
Variable	Any	Any	❌ Straggler: one GPU shows variance while others are steady

Prometheus Alert Rules for DCGM

```
# /etc/prometheus/rules/gpu_alerts.yml

groups:
  - name: gpu_health
    rules:
      - alert: GPUThermalThrottle
        expr: DCGM_FI_DEV_CLOCK_THROTTLE_REASONS > 0
        for: 5m
        labels: { severity: warning }
        annotations:
          summary: "GPU {{ $labels.gpu }} throttling on {{ $labels.Hostname }}"

      - alert: GPUXIDError
        expr: DCGM_FI_DEV_XID_ERRORS > 0
        for: 1m
        labels: { severity: critical }
        annotations:
          summary: "XID error on {{ $labels.Hostname }} GPU {{ $labels.gpu }}"

      - alert: GPUHighTemperature
        expr: DCGM_FI_DEV_GPU_TEMP > 83
        for: 5m
        labels: { severity: warning }

      - alert: GPUUncorrectableECC
        expr: DCGM_FI_DEV_ECC_DBE_VOL_TOTAL > 0
        for: 1m
```

```
      labels: { severity: critical }

  - alert: GPUSMUnderutilized
    expr: DCGM_FI_PROF_SM_ACTIVE < 50
    for: 10m
    labels: { severity: warning }
    annotations:
      summary: "SM utilization < 50% for 10min on {{ $labels.Hostname }}"
```

B.5 NCCL — Collective Communication Reference

> **Purpose:** Diagnose AllReduce bandwidth, communication topology issues, collective hangs, and NIC/InfiniBand degradation in multi-GPU and multi-node training clusters.

Essential NCCL Environment Variables

Variable	Recommended Value	Effect
`NCCL_IB_HCA`	`mlx5_0:1,mlx5_1:1,...,mlx5_7:1`	Pin NCCL to specific NICs for rail-aligned topology (critical for multi-rail IB)
`NCCL_IB_QPS_PER_CONNECTION`	`4`	4 QPs per NIC connection — improves NIC utilization
`NCCL_NET_GDR_LEVEL`	`SYS`	Enable GPUDirect RDMA — bypass CPU for GPU ↔ NIC transfers
`NCCL_NVLS_ENABLE`	`1`	Enable NVLink SHARP (H100 + Quantum-2+) — up to 2× AllReduce speedup
`NCCL_SHARP_ENABLE`	`1`	Enable IB SHARP for in-network compute on Quantum-2 switches
`NCCL_TIMEOUT`	`1800`	30-min watchdog (default 300s too aggressive for large checkpoints)
`NCCL_BUFFSIZE`	`8388608`	8 MB intermediate buffer (default 4 MB can limit throughput)
`NCCL_MIN_NCHANNELS`	`4`	Minimum communication channels per connection
`NCCL_DEBUG`	`WARN` (prod) / `INFO` (debug)	Logging verbosity — INFO mode has significant overhead
`NCCL_TOPO_DUMP_FILE`	`/tmp/nccl_topo.xml`	Dump topology XML for debugging GPU-NIC-Switch connectivity
`NCCL_ALGO`	Leave unset (auto)	Force ring/tree/collnet — only override if topology known suboptimal
`NCCL_SOCKET_IFNAME`	`^lo,^docker`	Exclude loopback and docker interfaces from NCCL

nccl-tests — Cluster Health Benchmark

```
# ── BUILD (one time) ──────────────────────────────────────────────
git clone https://github.com/NVIDIA/nccl-tests
cd nccl-tests
make MPI=1 MPI_HOME=/usr/local/mpi CUDA_HOME=/usr/local/cuda

# ── ALLREDUCE SWEEP (most important test) ─────────────────────────
mpirun -np 8 --bind-to none \
  ./build/all_reduce_perf \
  -b 1M -e 4G -f 2 \     # message sizes 1MB → 4GB, 2× steps
  -g 1                    # 1 GPU per process
```

```
# ── MULTI-NODE ALLREDUCE (8 nodes × 8 GPUs = 64 GPUs) ──────────────────────
mpirun -np 64 \
  -H node001:8,node002:8,...,node008:8 \
  --mca btl_openib_if_include mlx5_0:1 \
  ./build/all_reduce_perf -b 256M -e 4G -f 2 -g 1

# ── EXAMPLE HEALTHY OUTPUT ──────────────────────────────────────────────────
#   #                                                     out-of-place
#   #       size         count    type   redop    root    time   algbw   busbw
#   #        (B)    (elements)                               (us)  (GB/s)  (GB/s)
#    268435456      67108864   float     sum      -1    1423.7  188.5   188.5
#
# H100 × 8, NVLink healthy baseline:
#   256MB  busbw ≥ 160 GB/s
#   1GB    busbw ≥ 180 GB/s
#   ALERT: busbw < 140 GB/s → NVLink degradation → check DCGM field 1011/1012
#
# H100 × 64, IB NDR 400G healthy baseline:
#   256MB  busbw ≥ 40 GB/s
#   1GB    busbw ≥ 45 GB/s
#   ALERT: busbw < 35 GB/s → IB degradation → check ibstatus

# ── TOPOLOGY DEBUGGING ──────────────────────────────────────────────────────
NCCL_TOPO_DUMP_FILE=/tmp/topo.xml python -c \
  "import torch.distributed as dist; dist.init_process_group('nccl')"
cat /tmp/topo.xml
# Verify each GPU-i connects to same leaf switch (rail-optimized topology)

# ── ISOLATE HANG ────────────────────────────────────────────────────────────
# If training hangs, dump thread stacks to find stuck ranks:
py-spy dump --pid $(pgrep -f "train.py")
# Look for threads stuck in ncclAllReduce → rank is hanging
```

busbw vs algbw — Which Number Matters?

> **Use busbw (bus bandwidth) for cluster health assessment.** The `algbw` (algorithm bandwidth) measures bytes sent per second from the application's perspective. The `busbw` normalizes for the AllReduce algorithm's communication volume — for a ring AllReduce, busbw = algbw × 2(N-1)/N. For a healthy 8-GPU NVLink ring, you expect busbw ≈ algbw × 1.75. If busbw/algbw ratio is wrong, NCCL is using an unexpected algorithm.

B.6 vLLM Inference Metrics Reference

> **Purpose:** Monitor inference serving latency, throughput, KV cache pressure, and request queue depth. These metrics are the primary SLO health indicators for production LLM serving.

Start vLLM with Metrics Endpoint

```
python -m vllm.entrypoints.openai.api_server \
  --model meta-llama/Llama-3-70b-instruct \
  --tensor-parallel-size 2 \
  --quantization fp8 \
  --gpu-memory-utilization 0.90 \
  --enable-prefix-caching \
  --port 8000

# Pull metrics:
curl http://localhost:8000/metrics | grep vllm:
```

vLLM Metric Reference with Alert Thresholds

Metric	Type	Description	Alert Threshold
`vllm:gpu_cache_usage_perc`	Gauge	KV cache blocks used / total blocks	**> 0.82 → CASCADE RISK**
`vllm:cpu_cache_usage_perc`	Gauge	CPU offload cache utilization	> 0.90 → swap pressure
`vllm:num_requests_running`	Gauge	Requests currently in flight (prefill+decode)	Near max_num_seqs
`vllm:num_requests_waiting`	Gauge	Requests queued, not yet scheduled	> 50 → throughput limited
`vllm:time_to_first_token_seconds`	Histogram	Prefill phase latency per request	P99 > SLO threshold
`vllm:time_per_output_token_seconds`	Histogram	Per-token decode latency	P99 > 100ms typical
`vllm:e2e_request_latency_seconds`	Histogram	Full request latency (TTFT + decode)	P99 per SLO
`vllm:generation_tokens_total`	Counter	Total output tokens generated	Use `rate()[1m]` for throughput
`vllm:prompt_tokens_total`	Counter	Total input tokens processed	Rate tracks prefill load
`vllm:request_success_total`	Counter	Successful completions	Compare to request_count for error rate
`vllm:cache_config_info`	Gauge	KV cache configuration metadata	Informational
`vllm:spec_decode_draft_acceptance_rate`	Gauge	Speculative decoding acceptance rate α	< 0.65 → wrong draft model for this workload

🚨 **KV Cache Cascade:** When `vllm:gpu_cache_usage_perc` exceeds ~0.85, vLLM begins preempting (evicting) in-flight requests to reclaim KV blocks. Preemption causes requests to re-prefill from scratch — TTFT spikes 5–20× for affected requests. Alert at **0.82** to give headroom to react. Fix: reduce `max_num_seqs`, enable chunked prefill, or reduce `max_model_len`. See **Chapter 11, §11.3**.

B.7 PromQL Patterns for AI Cluster Monitoring

> **Purpose:** Production-ready PromQL queries for Grafana dashboards and Prometheus alert rules. Each query maps to a specific operational question.

GPU Compute Queries

```
# ── CLUSTER-WIDE SM UTILIZATION AVERAGE ──────────────────────────────
avg(DCGM_FI_PROF_SM_ACTIVE) by (Hostname)

# ── FIND UNDERUTILIZED GPUs (potential stragglers) ──────────────────
DCGM_FI_PROF_SM_ACTIVE < 50

# ── TENSOR CORE UTILIZATION PER GPU ─────────────────────────────────
DCGM_FI_PROF_PIPE_TENSOR_ACTIVE

# ── GPU POWER DRAW ACROSS FLEET ─────────────────────────────────────
DCGM_FI_DEV_POWER_USAGE

# ── MEMORY BANDWIDTH PRESSURE ───────────────────────────────────────
DCGM_FI_PROF_DRAM_ACTIVE > 0.95   # consistently near 100% = BW bottleneck
```

Thermal and Health Queries

```
# ── HOT GPUs ACROSS FLEET ───────────────────────────────────────────
DCGM_FI_DEV_GPU_TEMP > 80

# ── CLOCK THROTTLE ACTIVE ───────────────────────────────────────────
DCGM_FI_DEV_CLOCK_THROTTLE_REASONS > 0

# ── ECC CORRECTABLE ERROR RATE (rolling 1hr window) ─────────────────
rate(DCGM_FI_DEV_ECC_SBE_VOL_TOTAL[1h])

# ── ANY UNCORRECTABLE ECC (critical) ────────────────────────────────
DCGM_FI_DEV_ECC_DBE_VOL_TOTAL > 0

# ── ECC ERRORS PER NODE IN LAST 24H ─────────────────────────────────
sum by (Hostname) (increase(DCGM_FI_DEV_ECC_SBE_VOL_TOTAL[24h]))

# ── FREE GPU MEMORY (low = risk of OOM) ─────────────────────────────
DCGM_FI_DEV_FB_FREE < 5000    # in MiB
```

Inference Service Queries

```
# ── TTFT P99 LATENCY (5-minute window) ──────────────────────────────
histogram_quantile(0.99,
  rate(vllm:time_to_first_token_seconds_bucket[5m])
)

# ── TPOT P50 LATENCY ────────────────────────────────────────────────
histogram_quantile(0.50,
  rate(vllm:time_per_output_token_seconds_bucket[5m])
)

# ── CURRENT TOKENS/SECOND THROUGHPUT ────────────────────────────────
rate(vllm:generation_tokens_total[1m])

# ── KV CACHE UTILIZATION (alert if > 82%) ───────────────────────────
vllm:gpu_cache_usage_perc * 100 > 82

# ── REQUEST QUEUE DEPTH ─────────────────────────────────────────────
vllm:num_requests_waiting

# ── INFERENCE ERROR RATE ────────────────────────────────────────────
rate(vllm:request_success_total[5m]) / rate(vllm:request_count_total[5m])
```

Training Queries

```
# ── MFU PER TRAINING JOB ────────────────────────────────────────────
training_mfu

# ── MFU DROP DETECTION (alert if drops > 5 pp in 10 min) ────────────
training_mfu - training_mfu offset 10m > 0.05

# ── STEP TIME P99 (straggler detection) ─────────────────────────────
histogram_quantile(0.99, rate(training_step_seconds_bucket[5m]))

# ── ALLREDUCE LATENCY TREND ─────────────────────────────────────────
avg_over_time(allreduce_time_ms[5m])

# ── GRADIENT NORM SPIKE DETECTION ───────────────────────────────────
train_grad_norm > 10     # potential training instability
```

PromQL Pattern Reference

Pattern	PromQL Syntax	Use Case
Rate of counter	`rate(metric[5m])`	Errors/sec, tokens/sec from counters
Increase over window	`increase(metric[1h])`	ECC errors in last hour
Aggregate by label	`avg by (Hostname) (metric)`	Per-node averages across fleet
Percentile	`histogram_quantile(0.99, rate(hist_bucket[5m]))`	P99 TTFT from vLLM histogram
Threshold filter	`metric > threshold`	Find GPUs above temperature
Change detection	`metric - metric offset 10m`	Detect sudden MFU drops
Multi-label sum	`sum by (job, Hostname) (metric)`	Power per training job per node
Absent metric alert	`absent(metric)`	Alert if exporter stops scraping

B.8 AMD Tool Equivalents (MI300X / MI355X)

> **Context:** AMD MI300X clusters use a parallel toolchain. The conceptual workflow is identical — system timeline → kernel analysis → fleet health — but the commands differ. See **Chapter 17, §17.8** for AMD profiling details.

NVIDIA → AMD Tool Equivalence Map

NVIDIA Tool	→	AMD Equivalent
`nsys profile (system timeline)`		`rocprof --sys-trace / roctracer`
`ncu (kernel hardware counters)`		`omniperf (roofline + counters)`
`dcgmi dmon (fleet telemetry)`		`rocm-smi + amd-smi (or rocm_smi_exporter)`
`nvidia-smi dmon (quick status)`		`rocm-smi --showallinfo`
`torch.cuda.nvtx.range_push()`		`torch.cuda.nvtx (same API on ROCm)`
`Nsight Systems GUI (nsys-ui)`		`Radeon GPU Profiler (RGP)`
`NCCL (communications library)`		`RCCL (drop-in replacement)`
`CUDA Graphs (launch overhead)`		`HIP Graph (same concept, HIP API)`
`cuBLAS / cuDNN kernels`		`rocBLAS / MIOpen / hipBLAS`

Fig B-2. NVIDIA to AMD tool equivalence. The profiling workflow — system timeline, kernel analysis, fleet health — is identical; only the command names change.

AMD rocprof Quick Reference

```
# ── SYSTEM TIMELINE (equivalent to nsys) ──────────────────────────────
rocprof --sys-trace --output-dir /tmp/rocprof_out python train.py

# ── KERNEL STATISTICS ─────────────────────────────────────────────────
rocprof --stats python train.py
# Output: top kernels by wall time — equivalent to nsys stats gputrace

# ── HARDWARE COUNTERS ─────────────────────────────────────────────────
cat > /tmp/counters.txt << EOF
SQ_WAVES
GRBM_COUNT
TCC_HIT_sum
TCC_MISS_sum
EOF
rocprof --counters /tmp/counters.txt python train.py

# ── OMNIPERF ROOFLINE ANALYSIS ────────────────────────────────────────
omniperf profile -n my_run -- python train.py
omniperf analyze -p workloads/my_run/MI300_A1 --roof-only

# ── AMD-SMI FLEET HEALTH ──────────────────────────────────────────────
amd-smi monitor -p -t -u -m    # power, temp, utilization, mem
rocm-smi --showallinfo          # comprehensive single-GPU status
```

B.9 Storage and I/O Diagnostic Commands

> **Purpose:** Diagnose checkpoint I/O bottlenecks, dataset loading stalls, and parallel filesystem (Lustre/GPFS) health. Storage is the silent killer of MFU — a healthy training job loses 2–8% MFU to checkpoint overhead; a sick storage system can stall an entire cluster.

Checkpoint I/O Diagnostics

```
# ── MEASURE ACTUAL CHECKPOINT WRITE SPEED ─────────────────────────────
# Time a checkpoint save directly:
time python -c "
import torch
model = ... # your model
torch.save(model.state_dict(), '/path/to/checkpoint/ckpt.pt')
"
# A 70B BF16 model checkpoint = ~140 GB
# Expected: NFS 10GbE = ~60 sec; Lustre 200 GB/s = ~0.7 sec

# ── FIO SEQUENTIAL BENCHMARK (measure storage peak throughput) ────────
```

```
# Sequential write (checkpoint scenario):
fio --name=seqwrite --ioengine=libaio --direct=1 \
    --rw=write --bs=4M --numjobs=16 \
    --size=10G --runtime=60 --group_reporting \
    --filename=/mnt/scratch/testfile
# Target for Lustre with 10+ OSS nodes: > 50 GB/s aggregate write

# Sequential read (dataset prefetch scenario):
fio --name=seqread --ioengine=libaio --direct=1 \
    --rw=read --bs=1M --numjobs=32 \
    --size=10G --runtime=60 --group_reporting \
    --filename=/mnt/scratch/testfile

# ── LUSTRE FILESYSTEM HEALTH ──────────────────────────────────────
lfs df -h /mnt/scratch           # Capacity per OST
lfs df -i /mnt/scratch           # Inode usage per MDT
lfs check servers                 # OSS/MDS connectivity
lfs getstripe /path/to/file      # File stripe layout
lfs setstripe -c 64 /path/to/dir # Set 64-OST stripe for large files

# Lustre stats:
cat /proc/fs/lustre/llite/*/stats  # Client I/O statistics

# ── GPFS (IBM Spectrum Scale) HEALTH ──────────────────────────────
mmlsfs gpfs0 -T         # Filesystem configuration
mmfsck gpfs0 -n         # Check without repair
mmpmon fs_io_s          # I/O statistics

# ── IDENTIFY DATA LOADING BOTTLENECKS IN PYTHON ────────────────────
# Add instrumentation to DataLoader:
import time

class TimedDataLoader:
    def __init__(self, loader):
        self.loader = loader
        self.wait_times = []

    def __iter__(self):
        t0 = time.perf_counter()
        for batch in self.loader:
            wait = time.perf_counter() - t0
            self.wait_times.append(wait)
            yield batch
            t0 = time.perf_counter()

# If avg_wait_ms > 10ms → data loading is bottlenecking GPU

# ── NETWORK STORAGE (NFS) TUNING ──────────────────────────────────
# Mount options for checkpoint performance:
# nfs4 rsize=1048576,wsize=1048576,hard,intr,rw,async
# Check current NFS stats:
nfsstat -c              # Client statistics
mountstats /mnt/nfs     # Detailed per-mount stats
```

Storage Health Checklist

Symptom	Diagnostic Command	Expected vs Alert
Checkpoint saves taking > 5 minutes	`fio --rw=write --bs=4M --numjobs=16`	> 10 GB/s healthy; < 2 GB/s → storage degraded
Dataset loading stalls between steps	`nsys --trace=osrt` + CPU idle gaps	Gaps < 5ms fine; > 50ms → increase DataLoader workers
Lustre OST space imbalance	`lfs df -h`	All OSTs should be within 20% of each other
Inode exhaustion (small file datasets)	`lfs df -i`	> 80% inode usage → pack small files into webdataset / TAR
Random read latency for checkpointing	`fio --rw=randread --bs=4k`	HDD: < 500 IOPS critical; SSD: > 100K IOPS expected

B.10 Symptom-to-Command Quick Reference

This is the highest-frequency lookup table in this appendix — use it when you have a symptom but haven't yet identified the tool.

Symptom → Command → First Metric Quick Reference

SYMPTOM	COMMAND	FIRST METRIC
Low MFU in training (< 40%)	`nsys --trace=cuda,nvtx,nccl`	Timeline: idle gaps between kernels
High TTFT P99 in inference	`vLLM metrics: time_to_first_token P99`	Prefill kernel duration vs. decode
AllReduce bottleneck suspected	`NCCL_DEBUG=INFO + nccl-tests`	busbw vs. healthy baseline
Specific kernel is slow	`ncu --set full --kernel-name X`	Roofline position + Tensor Core %
GPU utilization looks low	`dcgmi dmon -e 1003,1004,1005,1009`	SM_ACTIVE (field 1003), not GPU_UTIL
GEMM underperforming	`ncu --metrics pipe_tensor matmul`	Tensor Core utilization % (target > 60%)
Straggler node suspected	`dcgmi dmon -e 1003 across nodes`	Outlier SM_ACTIVE vs. rest of fleet
Data loading bottleneck	`nsys --trace=cuda,osrt`	CPU idle gaps between CUDA launches
NVLink underperformance	`dcgmi dmon -e 1011,1012`	NVLink TX/RX bytes/s vs. 450 GB/s
IB link degradation	`ibstatus + nccl-tests busbw`	busbw vs. 45 GB/s (64-GPU NDR baseline)
ECC error escalation	`nvidia-smi --query-gpu=ecc.errors`	Corrected ECC count trend / hour
Thermal throttling [CRITICAL]	`dcgmi dmon -e 100,252`	Temp > 83°C + field 252 > 0
KV cache cascade risk	`curl localhost:8000/metrics | grep cache`	gpu_cache_usage_perc > 0.82 CRITICAL
Checkpoint I/O too slow	`fio --rw=write --bs=4M --numjobs=16`	Sequential write MB/s vs. Lustre baseline
Training hang / NCCL deadlock	`py-spy dump --pid $(pgrep python)`	Thread stack: stuck in ncclAllReduce?

Fig B-3. Symptom → command → first metric lookup table. Print and post at your workstation. Every row is an entry point into a deeper diagnostic workflow.

Tool Landscape Overview

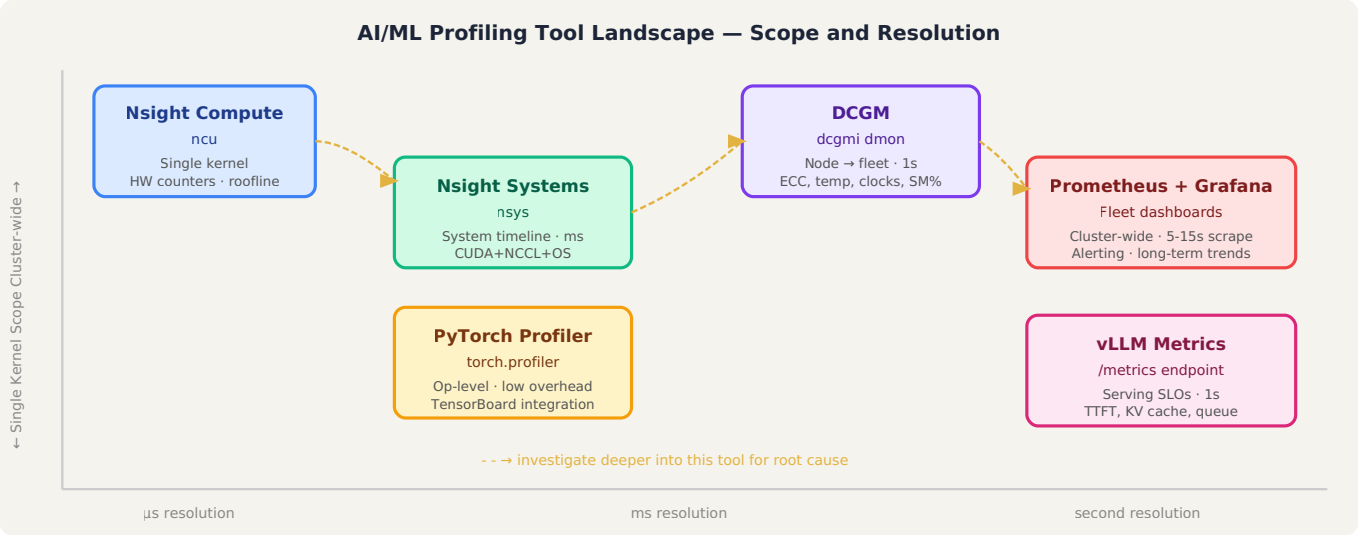


Fig B-4. Profiling tool landscape by diagnostic scope and time resolution. The typical workflow: Prometheus detects an anomaly → nsys identifies where time is lost → ncu diagnoses the specific kernel.

Cross-Chapter References

Topic	This Appendix Section	Deep Coverage
Nsight Systems workflow and NVTX markers	B.2	Chapter 17, §17.2
Nsight Compute and roofline interpretation	B.3	Chapter 17, §17.3 · Chapter 1, §1.4
DCGM architecture and Prometheus integration	B.4	Chapter 17, §17.2 and §17.4
NCCL ring AllReduce and topology	B.5	Chapter 14, §14.3–14.5
vLLM KV cache and cascade eviction	B.6	Chapter 11, §11.3
PromQL and Grafana dashboard design	B.7	Chapter 17, §17.4 and §17.5
AMD profiling toolchain (omniperf, rocprof)	B.8	Chapter 17, §17.8
Checkpoint I/O and storage architecture	B.9	Chapter 15, §15.3
MFU calculation and interpretation	B.10	Chapter 1, §1.5 · Chapter 12