PART IV · CHAPTER 12

Benchmarking & Performance Methodology

Measuring Correctly Before Optimizing Anything — From Microbenchmarks to Cluster-Scale Profiling

Book: AI/ML Infrastructure from Silicon to Scale | **Author:** Venkat Vinjam | **Cross-ref:** *GPU kernels (Ch 7) · KV cache (Ch 11) · Speculative decoding (Ch 13) · Networking (Ch 14) · Telemetry (Ch 17)*

> *"The most dangerous number in AI infrastructure is an unverified benchmark. It misleads architecture decisions, justifies wrong hardware purchases, and hides real problems behind confident-sounding percentages. Measure correctly first. Then optimize."*
>
> — AI Performance Engineering Principle

Every optimization decision in this book — kernel fusion, quantization, tensor parallelism, speculative decoding — produces a number. Whether that number is trustworthy depends entirely on how it was measured. A 40% TFLOPS improvement on a single kernel means nothing if the system is memory-bandwidth bound. A 1.8× throughput gain in a microbenchmark means nothing if it does not survive the production serving stack.

This chapter provides the complete benchmarking and measurement methodology for a Principal Performance Architect: measurement validity rules, the Roofline model as diagnostic framework, MLPerf interpretation, GPU microbenchmark suite, LLM-specific benchmarks, MFU measurement, a five-step profiling decision tree, Nsight Compute at the principal workflow level, and regression CI pipeline design.

FOUNDATIONAL

- Name the three physical conditions for a valid GPU measurement
- Explain why wall-clock time is wrong for GPU benchmarking
- Understand the Roofline ridge point and optimization direction
- Know the three MLPerf inference scenarios and their metrics

INTERMEDIATE

- Write a CUDA-event-timed benchmark with warmup and percentile reporting
- Classify any LLM operation as compute- or memory-bound using AI formula
- Run the GPU microbenchmark suite and interpret all five results
- Build the throughput-latency curve and identify the knee

PRINCIPAL LEVEL

- Apply the 5-step tree to diagnose any underperforming system
- Design a three-tier regression CI pipeline with statistical thresholds
- Critique MLPerf results for hardware procurement decisions
- Specify vendor acceptance criteria for a new 4,096-GPU cluster

12.1 — Why Benchmarking Is Hard

12.1.1 The Three Physical Realities

Warmup. GPU hardware does not operate at peak from cold state: the power management unit ramps clocks over 100–500ms, the CUDA JIT compiles kernels on first execution, and driver page tables must be populated. Any benchmark timing from iteration 1 captures initialization, not steady-state performance.

```
WARMUP BUDGET BY OPERATION TYPE:
  Single kernel microbenchmark:     5–10 iterations minimum
  GEMM throughput benchmark:        10–20 iterations (JIT + clock ramp)
  LLM prefill benchmark:            3–5 full forward passes
  LLM decode benchmark:             20–50 decode steps
  Training step benchmark:          5–10 steps (optimizer state warmup)
  Multi-GPU training:               10–20 steps (NVLink/IB calibration)
Rule: allocate max(10 kernel runs, 30 seconds wall clock)
```

Synchronization. CUDA launches are asynchronous: `kernel_launch()` returns immediately after enqueueing. Wall-clock timing around a kernel call measures host-side overhead, not GPU execution time.

```
CUDA SYNCHRONIZATION — CORRECT vs WRONG:

# WRONG: measures host-side enqueue time only (~10–50 µs)
t0 = time.perf_counter()
output = model(input)
t1 = time.perf_counter()    # MEANINGLESS for GPU work

# CORRECT: CUDA events give µs-precision GPU timing
start = torch.cuda.Event(enable_timing=True)
end   = torch.cuda.Event(enable_timing=True)
for _ in range(10): _ = model(input)   # warmup
torch.cuda.synchronize()
start.record()
output = model(input)
end.record()
torch.cuda.synchronize()
latency_ms = start.elapsed_time(end)   # µs-precision
```

Steady Thermal State. Under sustained load, die temperature rises until the power management controller throttles clocks. A 10-second benchmark may capture the pre-throttle peak; a 60-second run may show 8–15% lower throughput. Only the sustained number is relevant for production planning.

```
THERMAL PROTOCOL:
  Pre-benchmark: gpu-burn for 5 min → verify ±5W power draw, ±2°C temp stability
  During: monitor DCGM_FI_DEV_CLOCKS_THROTTLE_REASONS
  Post: if first-10s vs last-10s differ >5%: report the sustained number
```

12.1.2 The Five Common Measurement Mistakes

#	Mistake	Typical Error	Fix
1	No warmup — timing first forward pass	1.5–3× slower than steady state	Minimum 10 iterations before timing
2	Wrong batch size vs production	Memory-bound → compute-bound regime shift; vendor claims 3× higher tok/s at B=512 irrelevant for B=4 production	Benchmark at production batch distribution
3	Wrong precision (FP8 bench for BF16 prod)	2× throughput difference on H100 TC	Use minimum precision meeting quality target
4	Wall clock instead of GPU time	For <100μs kernels: error exceeds kernel time	CUDA events for kernel-level; `nsys` for system
5	Single run or mean only — ignoring variance	P99 can be 2–6× above P50; mean hides the tail cliff	≥100 measurements; report P50/P95/P99

12.1.3 Statistical Rigor: Why Percentiles Matter

```
EXAMPLE DECODE LATENCY DISTRIBUTION (70B, H100, batch=32):
  P50: 18.2ms | P90: 21.3ms | P95: 24.7ms | P99: 41.8ms | P99.9: 112.4ms | Mean: 19.9ms

  Mean (19.9ms) HIDES the P99 cliff at 41.8ms.
  At 100 RPS: ~1 request/second experiences >41.8ms; ~1/10s experiences >112.4ms.
  SLA contract is with the TAIL, not the average.
```

12.1.4 Noise Sources

Source	Magnitude	Detection	Mitigation
Thermal throttling	5–15%	`nvidia-smi --query-gpu=clocks_throttle_reasons`	Steady-state warmup; report sustained clock
NUMA effects	1.5–2× PCIe latency	`numactl --hardware`	`numactl --cpunodebind=0 --membind=0`
CPU frequency scaling	2–5× dispatch overhead	`cpupower frequency-info`	`cpupower frequency-set --governor performance`
Memory allocation spikes	100µs–10ms	Spike pattern in per-iteration timing	Pre-allocate; `torch.cuda.empty_cache()`
OS scheduler interrupts	50–500µs jitter	Irregular CPU activity in nsys	`isolcpus` kernel param; `taskset`

12.2 — The Roofline Model as Diagnostic Framework

```
ROOFLINE FUNDAMENTALS:

  Arithmetic Intensity (AI) = FLOPs / Memory Traffic   [FLOP/byte]
  Ceiling: T ≤ min(Peak_FLOPS,  AI × Peak_BW)
  Ridge point: AI_ridge = Peak_FLOPS / Peak_BW

H100 SXM5:  989 TFLOPS BF16  /  3.35 TB/s  = 295 FLOP/byte ridge point

DECISION RULE:
  AI < 295 → MEMORY-BW BOUND: fuse ops, quantize data, increase batch
  AI > 295 → COMPUTE BOUND:   FP8 (2× compute roof), Tensor Core alignment

Most common error: applying FP8 compute to a memory-bound workload.
```

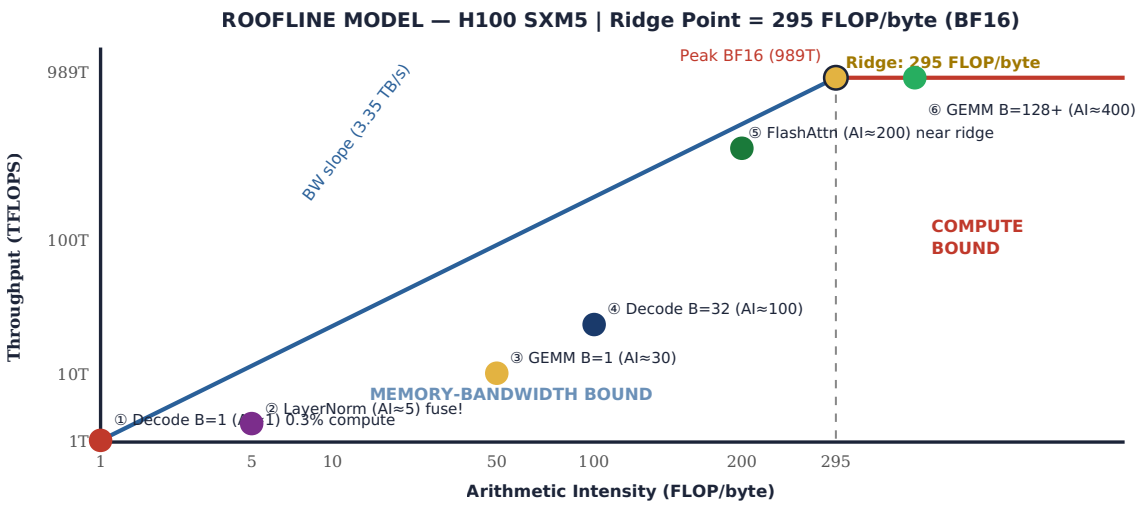


Figure 12.1 *— H100 SXM5 Roofline (log-log). Ridge point = 295 FLOP/byte. Operations left of the ridge are memory-bandwidth bound: the correct fix is fusion, quantization of data movement, or larger batch. Operations right of the ridge are compute-bound: the correct fix is FP8 (doubles compute roof) or Tensor Core alignment. Decode at B=1 sits at AI≈1, a full 295× below the ridge — it is HBM-bandwidth bound by an overwhelming margin.*

12.3 — MLPerf: The Industry Standard

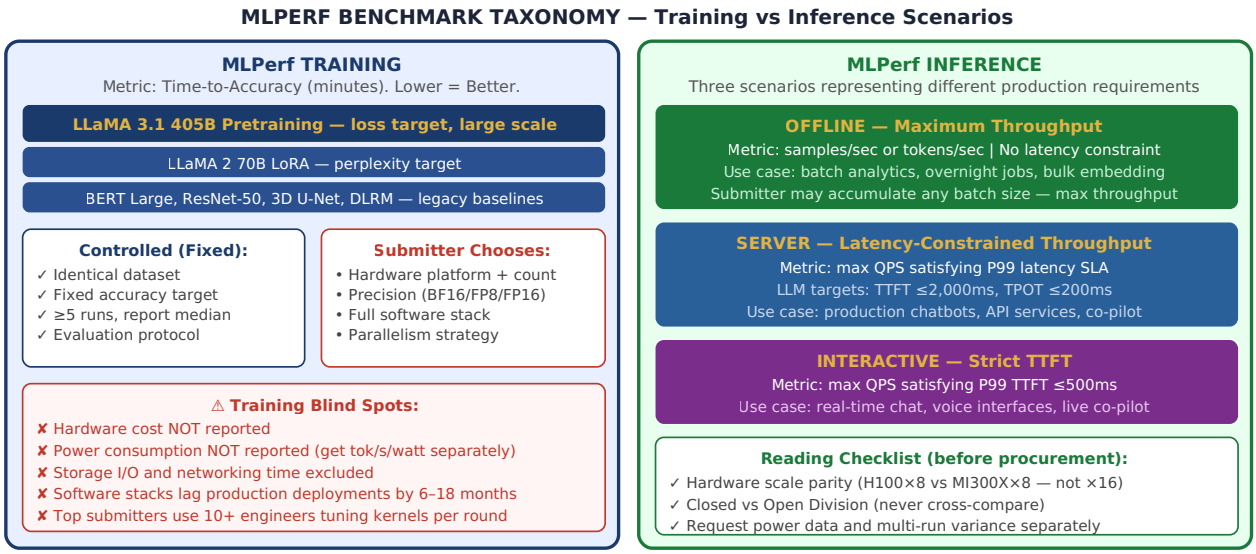


Figure 12.2 *— MLPerf taxonomy. Training measures time-to-accuracy (minutes, lower better). Inference defines three scenarios: Offline (unconstrained throughput), Server (max QPS within P99 latency SLA), Interactive (strict TTFT ≤500ms). Key blind spots: no power reporting, no real serving overhead, submitter-chosen software stacks 6–18 months ahead of production.*

12.4 — GPU Microbenchmark Suite

Benchmark	Tool	H100 SXM5 Target	Red Flag
GEMM throughput (BF16)	`cutlass_profiler`	800–900 TFLOPS (>80% of 989T)	<700T → Tensor Core issue
HBM bandwidth	`bandwidthTest` / STREAM	2,850–3,100 GB/s (>85% of 3,350)	<2,500 → ECC overhead or HBM error
NVLink BW (p2p)	`p2pBandwidthLatencyTest`	>400 GB/s per GPU pair	<200 → link failure; `nvlink --status`
Kernel launch overhead	Null-kernel loop (10K launches)	<5 µs (PyTorch); 1–3 µs (CUDA Graphs)	>10 µs → CPU governor issue
AllReduce BW	`nccl-tests` `all_reduce_perf`	>150 GB/s bus BW (8-GPU NVLink); >40 GB/s (64-GPU IB)	<80 GB/s (NVSwitch) → topology fault

12.5 — LLM-Specific Benchmarks

```
PREFILL (TTFT vs Sequence Length) — reveals operating regime:

Phase 1 — Overhead-bound (short seq, batch=1):
  TTFT flat floor = kernel launch + scheduling + tokenization
  Fix: CUDA Graphs, torch.compile()

Phase 2 — Compute-bound (medium–long seq, batch>1):
  TTFT scales linearly with seq_len. Desired: Tensor Cores utilized.

Phase 3 — BW-bound (very long seq, batch=1):
  Quadratic attention; Fix: FlashAttention or prefix caching

DECODE (TPOT vs Batch Size) — reveals HBM BW floor:
  batch=1 70B BF16: TPOT ≈ 42ms minimum (140 GB / 3,350 GB/s)
  INT4/FP8: halves model size → 1.8–2.2× TPOT improvement
```

12.5.3 Throughput-Latency Knee Curve

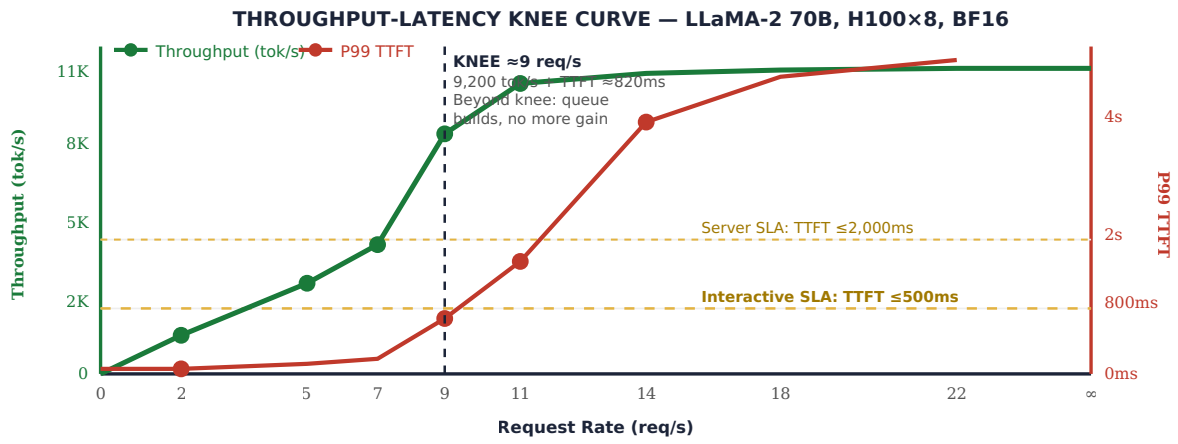


Figure 12.3 *— Throughput-latency knee curve (LLaMA-2 70B, H100×8). Throughput rises steeply to ~9 req/s then plateaus at ~11,000 tok/s. P99 TTFT stays stable until the same knee, then explodes as queues build. Sustainable operating points: 5–6 req/s (TTFT≤500ms interactive SLA), 10–11 req/s (TTFT≤2,000ms server SLA). Beyond the knee, adding requests increases queue depth but not throughput.*

12.6 — Training Benchmarks and MFU

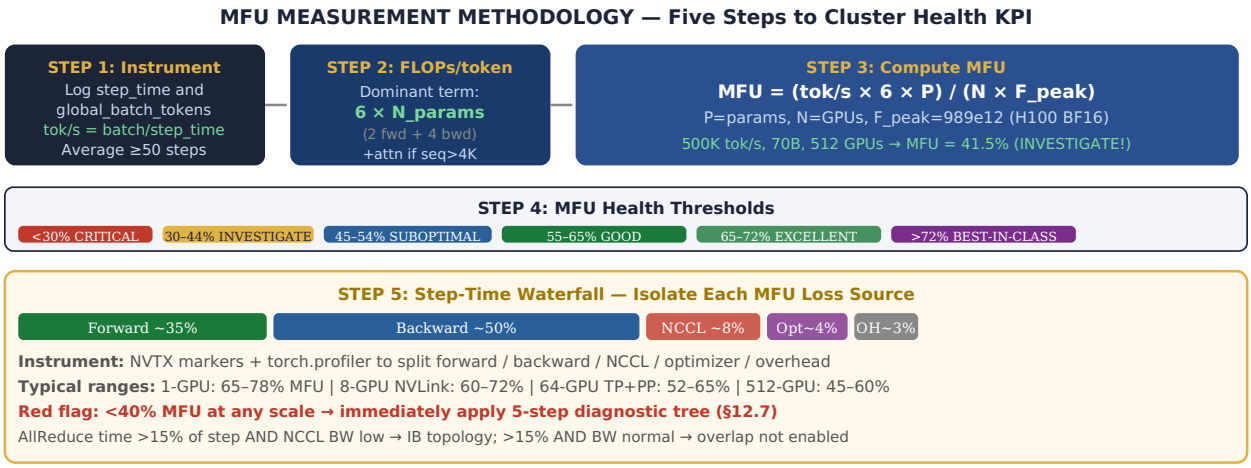


Figure 12.4 *— MFU measurement in five steps. Instrument (log tok/s), compute FLOPs/token (6×P), calculate MFU = (tok/s × 6P)/(N × F_peak), compare to health thresholds (<30% critical, 55–65% good, >72% best-in-class), and decompose the step-time waterfall using NVTX markers to isolate each loss source.*

12.7 — Five-Step Profiling Decision Tree

This tree converts an ambiguous performance symptom into a targeted action in 10–20 minutes using three DCGM counters and one system-level timer. Run nsys first, ncu only after all five steps.

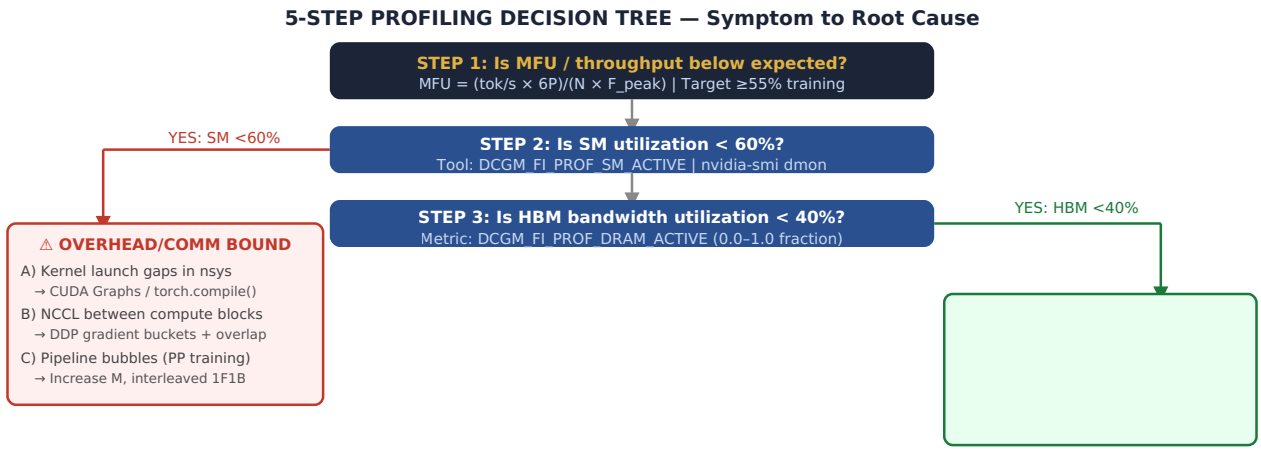


Figure 12.5 *— Five-step profiling decision tree. Uses three DCGM counters (SM_ACTIVE, DRAM_ACTIVE) and one system-level timer (AllReduce fraction from nsys). Each branch provides a specific root cause and fix. Only escalate to Nsight Compute after all five steps are exhausted. The tree can be traversed in 10–20 minutes on any underperforming system.*

12.8 — Nsight Compute: The Principal Architect Workflow

Nsight Compute (ncu) is the kernel-level profiler — it attributes performance to specific bottlenecks within one kernel execution. Use it only after Nsight Systems has confirmed the bottleneck is in a compute kernel, not in communication or scheduling overhead. Opening Nsight Compute without first running Nsight Systems is the single most common time-wasting mistake in GPU performance engineering.

```
PROFILER SELECTION RULE:

  QUESTION                          TOOL    SCOPE
  ──────────────────────────────────
  Where does time go in the step?   nsys    System timeline (ms-µs granularity)
  Which kernels take the most time? nsys    Kernel execution timeline
  Why is a specific kernel slow?    ncu     Within-kernel hardware counters
  Is AllReduce overlapped w/ backward? nsys Cross-stream NCCL timeline
  What is the roofline position?    ncu     Per-kernel roofline chart
  Are Tensor Cores utilized?        ncu     Pipe utilization counters
  Is there a regression vs baseline? both   ncu diff + nsys comparison

  Rule: nsys first, always. ncu only on the confirmed bottleneck kernel.
```

```
THE 5 METRICS THAT MATTER IN NSIGHT COMPUTE:

ncu --metrics \
  sm__active_cycles_avg.pct,                                    \
  sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_elapsed,  \
  gpu__dram_throughput.avg.pct_of_peak_sustained_elapsed,       \
  l2cache__hit_rate.pct,                                        \
  gpu__compute_memory_throughput.avg.pct_of_peak_sustained_elapsed  \
  --kernel-name "regex:gemm|attn|flash"                         \
  python benchmark.py

① SM Active %   >80% target   | <60%: occupancy problem
                                   Fix: reduce register/SMEM per thread, smaller tiles

② Tensor Core % >70% target   | <40%: alignment or data-feed bottleneck
                                   Fix: M,N,K dims divisible by 64 (BF16) / 128 (INT8)

③ HBM BW %      Regime-dependent:
                   BW-bound kernel:  >85% = excellent utilization
                   Compute-bound:    <40% = expected and correct
                   Red flag: 40–70% in a BW-bound kernel → access pattern inefficiency

④ L2 Hit Rate   >40% for kernels with reuse (attention, tiled GEMM)
                   <20%: streaming access — normal for pure streaming kernels
                   Low for attention: block size too small → increase tile, use FlashAttn

⑤ Roofline      Dot left of ridge → BW-bound; right → compute-bound
                   Below BW slope:    access inefficiency (uncoalesced, strided)
                   Below compute roof: register pressure, warp divergence
```

```
KERNEL REGRESSION DETECTION WORKFLOW:

# Step 1: Profile baseline (before change)
ncu --metrics sm__active_cycles_avg.pct,                              \
             sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_elapsed, \
             gpu__dram_throughput.avg.pct_of_peak_sustained_elapsed   \
    --export baseline_profile                                         \
    python benchmark.py --kernel flash_attention_v2

# Step 2: Apply change (new kernel / driver / library version)

# Step 3: Profile candidate
ncu --metrics [same] --export candidate_profile python benchmark.py

# Step 4: Generate diff report
ncu --diff baseline_profile.ncu-rep candidate_profile.ncu-rep --page details

# Regression signals in diff:
#   SM Active % decreased >5%       → new kernel has occupancy regression
#   Tensor Core % decreased >5%     → TC utilization regressed
#   HBM BW increased (BW-bound kern) → more data movement than before
#   Duration increased >3%          → throughput regression confirmed
```

12.9 — Building a Performance Regression CI Pipeline

Manual benchmarking before and after each change is insufficient at scale. A Principal Architect designs the performance regression CI pipeline as a first-class system component, not an afterthought. It is the infrastructure that protects months of optimization work from being silently erased by a driver update.

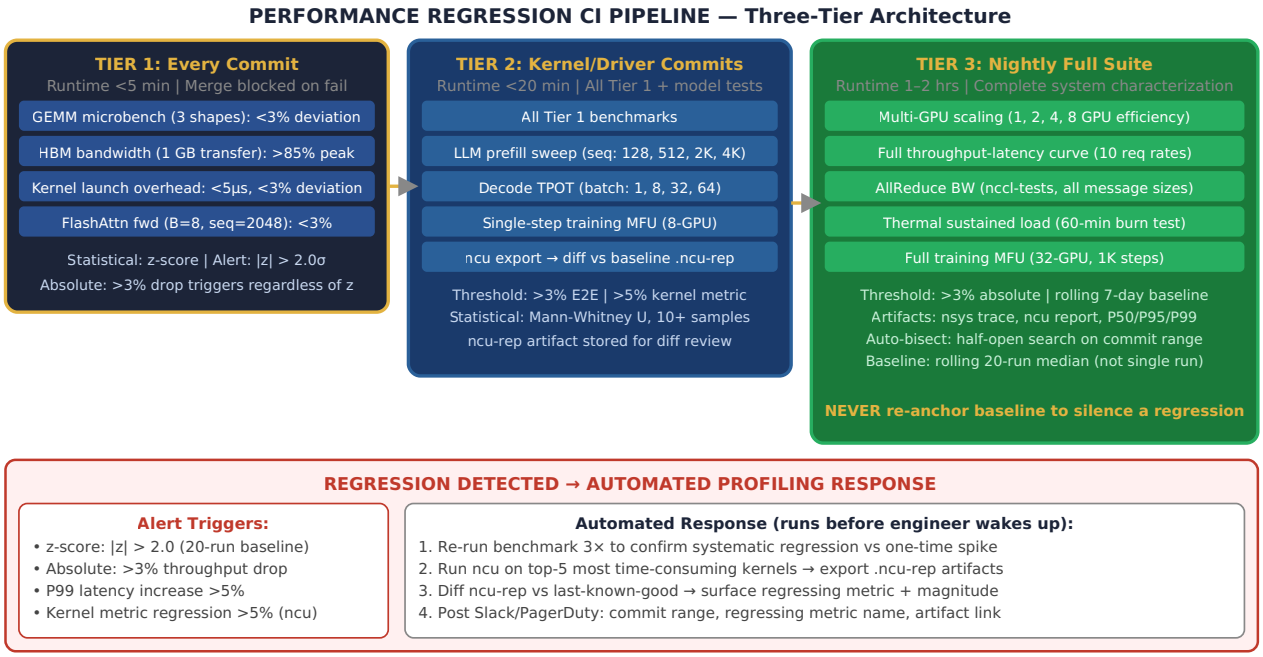


Figure 12.6 *— Three-tier performance regression CI pipeline. Tier 1 (every commit, <5 min): four core microbenchmarks gating merges. Tier 2 (kernel/driver commits, <20 min): model-level LLM benchmarks plus automated ncu diff vs stored baseline. Tier 3 (nightly, 1–2 hrs): full system characterization with auto-bisect. On any regression trigger: automated three-shot reconfirmation, ncu kernel profiling, diff report, and Slack alert with specific regressing metric — before the engineer is notified.*

```
STATISTICAL ALERT THRESHOLDS:

z-SCORE METHOD (primary):
  z = (today_run - rolling_baseline_mean) / baseline_std_dev
  Baseline: rolling 20-run median (not a single golden run)

  |z| > 2.0  → P0.95 confidence of regression → ALERT (Slack message)
  |z| > 3.0  → P0.997 confidence             → P1 PAGE (pagerduty)
  |z| < 1.5  → noise; do not alert

ABSOLUTE THRESHOLD (belt-and-suspenders):
  Also alert if throughput drops >3% vs baseline mean, even when z < 2.0
  Also alert if P99 latency increases >5% vs baseline median

BASELINE GOVERNANCE:
  Re-anchor baseline after: confirmed optimization PR, hardware change, monthly reset
  NEVER re-anchor to fix an unexplained regression — that converts detection into documentation
```

CHAPTER 12 QUICK REFERENCE — H100 SXM5 NUMBERS TO KNOW COLD

Metric	H100 SXM5 Value	Notes / Production Achieved
BF16 GEMM peak	989 TFLOPS	Achieved: 800–900T (80–90%). With sparsity: 1,979T.
FP8 GEMM peak	1,979 TFLOPS	2× BF16 compute roof; with sparsity: 3,958T
HBM3 bandwidth	3,350 GB/s (3.35 TB/s)	Achieved decode: 2,850–3,100 GB/s (85–92%)
NVLink 4.0 BW	900 GB/s/GPU unidirectional	AllReduce bus BW (NCCL, 8-GPU): 150–200 GB/s
Ridge point (BF16)	~295 FLOP/byte	989 / 3.35 = 295. Below: BW-bound. Above: compute-bound.
Kernel launch overhead	3–8 µs (PyTorch)	1–3 µs with CUDA Graphs. >10µs: check CPU governor.
Decode TPOT floor (70B BF16)	~42 ms (batch=1)	140 GB / 3,350 GB/s = 41.8ms. Physical HBM BW floor.
Decode TPOT (INT4)	~20–23 ms	1.8–2.2× improvement from halving bytes/weight-load
Prefill TTFT (70B, seq=2048, B=1)	280–350 ms	H100×8, BF16; compute-bound phase
Peak offline throughput (70B)	~11,000 tok/s	H100×8, BF16, Offline MLPerf scenario
Max server QPS (TTFT≤500ms)	~8–10 req/s	70B, H100×8. Beyond knee: latency explodes, throughput flat.

MFU Range	Assessment	Typical Context
<30%	Critical — stop and investigate immediately	Major configuration error or hardware fault
30–44%	Apply 5-step decision tree now	Likely comm overhead, pipeline bubble, or dispatch overhead
45–54%	Suboptimal — worth optimization effort	Large-scale multi-node with moderate tuning
55–65%	Good — well-configured cluster	64–512 GPU TP+PP+DP production job
65–72%	Excellent — top-tier configuration	8-GPU NVLink DDP with gradient overlap
>72%	Best-in-class — single GPU or optimal small config	Single H100 compute-dominant workload

CI Threshold	Alert Type	Action
z-score |z| > 2.0	P2 Slack alert	Auto-rerun 3×; ncu export; diff vs baseline
z-score |z| > 3.0	P1 PagerDuty page	Same + immediate on-call response
Absolute >3% throughput drop	Slack alert regardless of z	Belt-and-suspenders for stable but shifted distributions
P99 latency increase >5%	Slack alert	Tail regression check; re-profile with nsys
ncu kernel metric drop >5%	Tier 2 gate failure	Block merge; diff ncu-rep to identify regressing counter

Key Takeaways

MEASUREMENT DISCIPLINE

- Three physical realities gate any valid GPU measurement: warmup (allow 10+ iterations for JIT and clock ramp), synchronization (CUDA events only — never Python wall clock), and steady thermal state (verify ±5W power stability before timing). Violating any one produces a number that may mislead every optimization decision downstream.
- P99 latency is the SLA-relevant number; mean latency is what the average user experiences — and half your users exceed the mean. For a 100 RPS service with a P99 tail at 41.8 ms, one request per second exceeds that tail. Always report P50/P95/P99 as the minimum acceptable statistical output. The five common benchmark mistakes — no warmup, wrong batch size, wrong precision, wall-clock timing, ignoring variance — account for the majority of incorrect results published in engineering reports and vendor whitepapers.
- MLPerf is the industry standard for hardware comparison but a poor proxy for production capacity planning. Its structural blind spots — no power reporting, no real serving stack overhead, no multi-tenant interference, software stacks 6–18

months ahead of production — make every result a best-case ceiling. Always request tokens/second/watt, verify Closed vs Open Division, and confirm hardware scale parity before using MLPerf to justify a procurement decision.

ROOFLINE AND MFU

- The Roofline ridge point for H100 BF16 is 295 FLOP/byte. Every operation below it is memory-bandwidth bound: fuse kernels, quantize data movement, or increase batch size. Every operation above it is compute-bound: use FP8 (doubles the compute roof), align Tensor Core dimensions, or increase SM occupancy. Applying a compute-bound optimization to a memory-bound workload — the most common error in AI performance work — produces zero improvement. The roofline tells you which resource to optimize before touching any code.
- MFU is the single cluster-health KPI for training: MFU = (tok/s × 6P) / (N × F_peak). Anything below 40% at any scale warrants immediate investigation with the 5-step decision tree. The step-time waterfall — forward / backward / NCCL / optimizer / overhead — decomposes where MFU is lost and directs remediation to the right layer. Typical production ranges: 65–78% for single GPU, 52–65% for 64-GPU TP+PP, 45–60% for 512-GPU 3D-parallel.

DECISION TREES AND CI INFRASTRUCTURE

- The five-step profiling decision tree converts any ambiguous performance symptom into a targeted action in 10–20 minutes using three DCGM counters and one system-level timer. The sequence — MFU → SM utilization → HBM BW low? → HBM BW high? → AllReduce fraction — eliminates 80% of root cause possibilities before requiring Nsight Compute. Only escalate to ncu after all five steps are completed. Opening ncu without first traversing the tree is the most common profiling time-wasting mistake.
- A performance regression CI pipeline is not optional infrastructure — it is what prevents months of optimization from being silently erased by a driver update. The three-tier architecture gates merges at the right granularity: microbenchmarks on every commit (<5 min), model-level tests on kernel commits (<20 min), full system suite nightly (1–2 hrs). The baseline must be a rolling 20-run median with z-score alerting (|z|>2.0 for Slack, |z|>3.0 for page). Never re-anchor the baseline to silence an unexplained regression — that converts a detection system into a documentation system for undetected regressions.

Review Questions

FOUNDATIONAL Build the mental model

1. A colleague measures LLM inference latency using Python's `time.perf_counter()` and reports "our kernel runs in 0.8 ms." Explain exactly what this number actually measures, why it is incorrect for GPU kernel timing, and write the corrected measurement code using CUDA events with proper warmup and synchronization.

2. The H100 ridge point is 295 FLOP/byte. A decode operation at batch=1 has arithmetic intensity ≈1 FLOP/byte. (a) Is this operation compute-bound or memory-bound? (b) A team proposes switching from BF16 to FP8 compute to speed up decode. Explain using the roofline model exactly why this will not help. (c) What single change would actually improve decode TPOT at batch=1, and what is the expected improvement magnitude for LLaMA-3 70B on H100?

3. An MLPerf Inference result shows System A achieves 1.3× the throughput of System B on Llama-2 70B Offline. List three specific questions a principal architect must answer before this result justifies a hardware procurement decision. For each, explain what a concerning answer would look like.

INTERMEDIATE Apply and calculate

1. A training job runs LLaMA-3 70B on 256 H100s. You measure 420,000 tokens/sec globally. (a) Calculate the MFU. Is this in the acceptable range? (b) The 5-step tree shows: SM util = 82%, HBM BW = 72% of peak, AllReduce = 9% of step time. Which step surfaces the bottleneck and what is it? (c) What is the specific fix? (d) After fixing, you measure 530,000 tok/s. What is the new MFU?

2. You are building the throughput-latency curve for a 70B serving deployment on H100×8. At 6 req/s: 8,500 tok/s, P99 TTFT = 380 ms. At 10 req/s: 10,200 tok/s, P99 TTFT = 1,950 ms. At 14 req/s: 10,600 tok/s, P99 TTFT = 4,200 ms. (a) Identify the knee of the curve. (b) What is the maximum sustainable request rate for a P99 TTFT ≤500 ms SLA? (c) For a P99 TTFT ≤2,000 ms SLA? (d) A sales team proposes provisioning for 12 req/s to "maximize utilization." What specifically goes wrong at that operating point?

3. Your nightly CI runs decode throughput benchmarks. The 20-run baseline has mean 11,200 tok/s, std dev 180 tok/s. Tonight's run shows 10,650 tok/s. (a) Calculate the z-score and state whether this triggers an alert. (b) The absolute drop is 4.9%: does your 3% threshold trigger? (c) The automated ncu diff shows SM Active % dropped from 84% to 71% and Tensor Core % dropped from 76% to 61%. What does this pattern indicate, and which specific ncu metric would you examine next to distinguish register pressure from CUDA block size change?

PRINCIPAL LEVEL Design and defend

1. **FP8 benchmark credibility assessment:** An engineer reports that adding FP8 training improved throughput by 28% in a benchmark. Walk through the complete validation methodology: what three properties of the benchmark must you verify before trusting the number? What roofline analysis determines whether FP8 should yield any benefit for this specific workload? If the workload is compute-bound at AI = 400 FLOP/byte on H100, what is the theoretical maximum speedup from FP8, and why might 28% be plausible or implausible? What three questions would you ask the engineer?

2. **512-GPU cluster acceptance protocol:** Design the complete benchmark qualification protocol for accepting a 512-GPU H100 cluster from a vendor. Specify: (a) the five microbenchmarks run on every node individually with exact acceptance thresholds; (b) the inter-node test that identifies outlier links in the InfiniBand fabric; (c) the scaling efficiency test across all 64 node groups; (d) the decision rule when 3 of 64 groups fail the scaling test at >2σ below the mean. Justify each threshold against H100 SXM5 specifications.

3. **38% MFU root cause and remediation:** A 512-GPU training job runs at 38% MFU. SM utilization is 72%. HBM bandwidth is 68% of peak. AllReduce time is 22% of step time. (a) Walk through all five decision tree steps and identify which step surfaces the bottleneck. (b) Sub-diagnose: list the three sub-causes and the additional measurement that distinguishes them. (c) Specify the exact DDP/FSDP/NCCL configuration parameters you would change for the most likely

sub-cause. (d) Estimate the MFU improvement if the fix achieves 70% overlap efficiency — currently 0% — given T_backward = 900 ms and T_AllReduce = 780 ms. (e) What residual bottleneck do you investigate next after this fix?
