



CONFERENCE

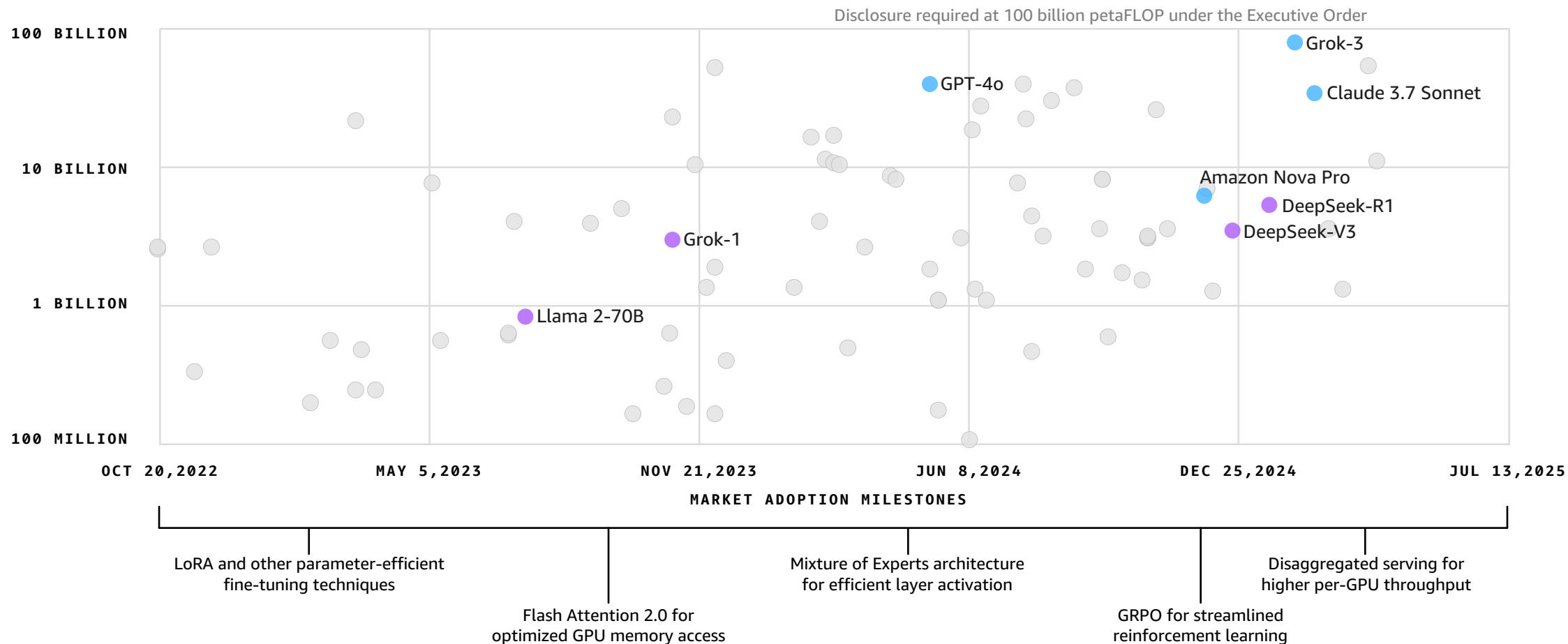
—NORTH AMERICA 2026—

Five Bottlenecks, Five Fixes: Systematic Performance Diagnosis for PyTorch Distributed Training

Aravind Neelakantan
Specialist Solutions Architect – GenAI
AWS

Paulo Aragao
Principal, Applied AI Architect
AWS

Foundation model compute demand outpaces FM development efficiency innovations

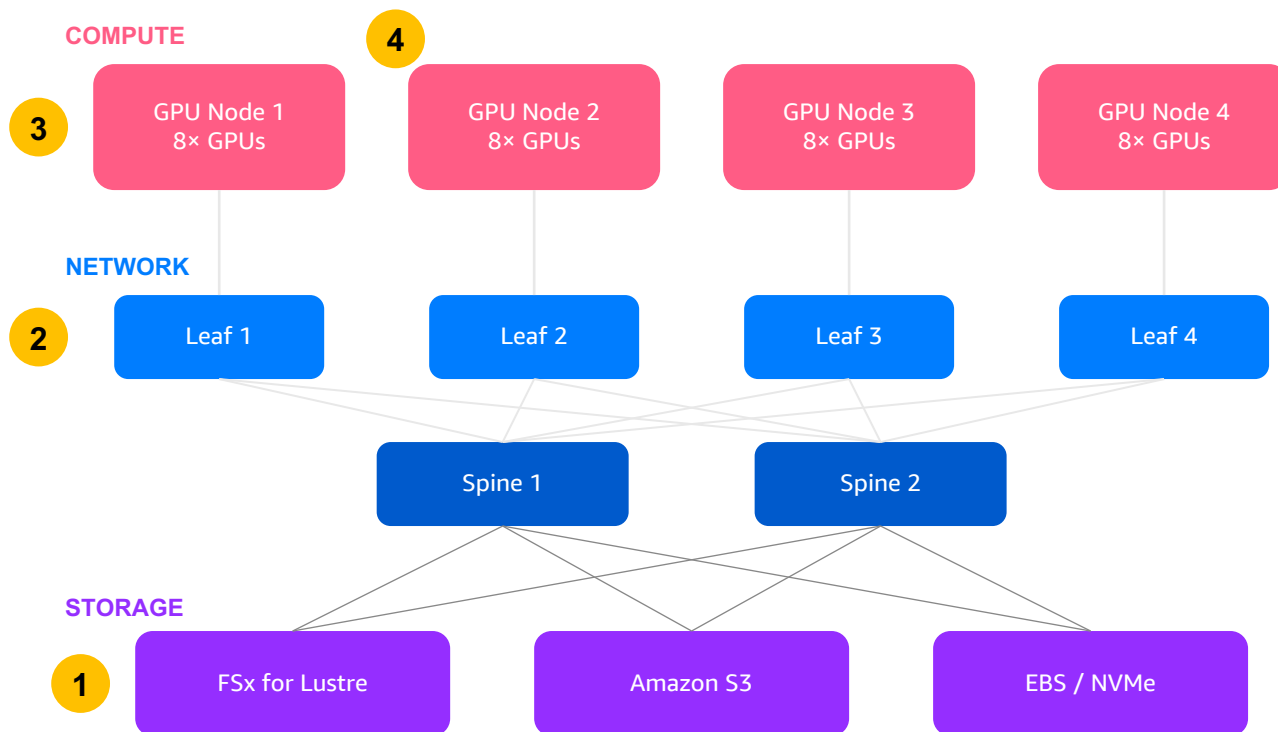


Data source: Epoch (2025)

© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved. Amazon Confidential and Trademark.

Distributed systems

The art of moving bottlenecks



1 Storage (I/O bound)

Data loading throughput, checkpoint writes.
Example: VJEPA video datasets are particularly I/O intensive.

2 Network (comm bound)

Collective ops: AllReduce, AllGather, All-to-All.
Dominates at scale with LLM training (e.g. Llama 70B+, Qwen3 30B and 235B).

3 Compute (CPU bound)

Data preprocessing, kernel launch overhead, CPU-GPU synchronization barriers.

4 GPU (compute bound)

FLOPS utilization, HBM bandwidth. Target state for optimal MFU and Tokens/second (throughput).

The diagnostic decision tree: name the bottleneck

If bound by...	torch.profiler / timeline signature	Counter / NCCL signature	First fix
Storage (I/O)	GPU idle at step start; data_time spikes (esp. epoch boundary)	sar -d await/aqu-sz; vmstat bi/bo; GPU idle gaps	mmap/indexed data, cycling sampler, prefetch, tune FSx/Lustre
Network (comm)	Kernels wait on ncclSendRecv / AllToAll; comm not overlapped	Low EFA util OR high msg count; 'Selected provider is efa'	Right-size PP/EP; keep EP $\leq$ GPUs/node; enable overlap
CPU (launch)	Gaps between kernels; host threads busy (sem_wait)	GPU-Util low with CPU pegged; kernel launch gaps	Fuse/boost kernels, CUDA graphs, more dataloader workers
GPU (compute)	Kernels back-to-back, SMs >99% busy, no gaps	DCGM 1002 SM active ≥ 0.8; 1004 Tensor active high	Kernel/attention backend (FA4), precision, MFU vs entitlement

Four layers: Storage · Network · CPU · GPU. Name it from the signature, then fix it.

Start with Hardware entitlement

Entitlement = what the instance can physically deliver, per resource. Read every benchmark result as a percentage of it; the gap is what tuning can still win.

How to calculate it

1 Compute = GPUs × peak **dense** TFLOPS at the precision you run. Datasheet Tensor Core peaks are quoted *with sparsity* (2× dense): halve them.^{1 2}

2 MFU (training) = achieved model TFLOPS per GPU ÷ peak dense TFLOPS. Model FLOPs ≈ 6 × active params × tokens, plus attention.³

3 HBM bandwidth (decode) tokens/s per stream ≤ HBM TB/s ÷ bytes read per token (weights + KV on each GPU). Low-batch decode is bandwidth-bound, not FLOP-bound.⁶

4 Network EFA Gbps ÷ 8 = GB/s per instance (3,200 Gbps = 400 GB/s); NVSwitch GB/s per GPU inside the node.⁴

	p5.48xlarge	p5en.48xlarge
GPU	8 × H100	8 × H200
BF16 dense / GPU	989 TFLOPS	989 TFLOPS
FP8 dense / GPU	1,979 TFLOPS	1,979 TFLOPS
HBM / GPU	80 GB · 3.35 TB/s	141 GB · 4.8 TB/s
NVSwitch / GPU	900 GB/s	900 GB/s
EFA / instance	3,200 Gbps (200 GB/s)	3,200 Gbps (400 GB/s)
CPU ↔ GPU PCIe	—	Gen5, up to 4× p5

Worked example — training

Llama-3.1-70B BF16, 64 × H200: entitlement 64 × 989.4 = 63.3 PFLOPS. Achieved 529.33 TFLOPS/GPU (33.9 PFLOPS) = 53.5% MFU.⁵

Worked example — inference decode

Gemma4-31B FP8, TP=4, conc 1, H200: ~7.75 GB of weights per GPU read every 8.86 ms ≈ 875 GB/s = ~18% of 4.8 TB/s. Headroom is in the kernels, not the hardware.⁶

[1] [NVIDIA H100 specs \(Tensor Core rows marked *\)](#) [2] [NVIDIA Hopper in-depth: sparsity doubles Tensor Core rate](#) [3] [AI Model Calculator, Phase 3 formula](#) [4] [AWS EC2 P5/P5e/P5en product details](#) [5] [Portal: pretrain-llama70b-bf16/p5en-8nodes, job 239](#) [6] [gemma4-31b-it benchmark repo, results/unified/RESULTS.md](#) [H200] [HBM 141 GB / 4.8 TB/s: nvidia.com/h200; compute peaks = portal's H200 denominators \(989.4 BF16, FP8 MFU 41.5% at 820.64 TFLOPS\)](#)



Which tool for which subsystem

Subsystem	Question	First look (low overhead, run always)	Deep dive
GPU compute	Are SMs and Tensor Cores busy?	nvidia-smi dmon -s puct . GPU-Util = % of time ≥ 1 kernel ran, not SM load. DCGM 1002 SM active (≥ 0.8 needed), 1004 Tensor active	nsys timeline + --gpu-metrics-devices
Clocks & power	Is the GPU throttling?	nvidia-smi -q -d PERFORMANCE (clock event reasons); dmon -s v power / thermal violations	Lock clocks for A/B runs: nvidia-smi -lgc
HBM	Capacity- or bandwidth-bound?	nvidia-smi --query-gpu=memory.used,memory.total ; DCGM 1005 DRAM active (~ 0.8 is the practical max)	nsys --cuda-memory-usage=true (adds overhead)
PCIe / topology	Is host  GPU staging the limit?	nvidia-smi dmon -s te (Rx/Tx MB/s, replays); DCGM 1009/1010; nvidia-smi topo -m GPU/NIC/CPU affinity	nsys GPU metrics: PCIe read/write % of peak; HtoD memcopy rows
NVLink / NVSwitch	Are intra-node collectives saturated?	DCGM 1011/1012 NVLink Tx/Rx bytes; nvidia-smi nvlink	nsys -t nccl + GPU metrics NVLink Rx/Tx
Network (EFA)	Is the fabric the limit, or unhealthy?	rdma -p statistic show : $\Delta\text{bytes} \div \Delta t$ vs 400 GB/s; retrans_* , impaired_remote_conn_events	nsys -t nccl ; CloudWatch EFA metrics
Storage	Is the data loader waiting on disk?	sar -d (await, aqu-sz); vmstat bi/bo	nsys -t gds (GPUDirect Storage)
CPU	Are cores or loader workers saturated?	sar -P ALL , sar -u ALL (%iowait, %steal); vmstat r	nsys -t osrt (OS runtime calls)
Host RAM & swap	Is the OS paging or reclaiming?	vmstat si/so; sar -W pswpin/s; sar -B majflt/s, pgscand/s	sar -H hugepages

Find the subsystem with always-on counters, then use nsys for attribution. Pause DCGM profiling while nsys runs.



Bottleneck 1 · I/O-bound: V-JEPA 2.1 Video Training

↓ How It Started

Model	V-JEPA 2.1 ViT-g (1B)
Cluster	64 × B200 GPUs
Iter time	4,980 ms (effective avg)
MFU	20.7% (effective)
Data loading	44% of iteration time
Epoch spikes	28s stalls every 16 steps

⚡ What Changed

Cycling sampler	Eliminates epoch stalls (-36.5%)
TorchCodec	data_time 140ms→24ms (-83%)
GradScaler noop	Remove FP16 overhead for BF16
FSDP	✗ 2× slower (model too small)
GPU decode	✗ 13-17× slower at boundaries
Lesson	Fast GPUs → data is the bottleneck

↑ How It Ended

Model	Same (DDP, BF16)
Cluster	64 × B200 GPUs
Iter time	3,071 ms
MFU	33.6%
Data loading	Eliminated as bottleneck
Epoch spikes	0 occurrences

39% faster iterations
Eliminated data loading bottleneck
Zero'ed epoch stalls

Bottleneck 2 · Communication/pipeline: Llama 3.1 70B FP8

↓ How It Started

Config	TP=4, PP=8 (as shipped)
Throughput	736.8 TFLOPS/GPU
MFU	~37% (est.)
GPU memory	~59 GB of 141 (lots spare)
Problem	PP=8 inherited from 16-GPU recipe
Step time	10.04 s / iteration

⚡ What Changed

PP 8→2	One parameter, no recipe edits
ncclSend	720 → 180 msgs/iter (4.0x)
Bubble	2.19% → 1.25% (NOT the cause)
EFA bytes	fewer, larger P2P transfers
Root cause	Msg COUNT, not pipeline bubble
Lesson	Inherited parallelism = #1 killer

↑ How It Ended

Config	TP=4, PP=2 (one change)
Throughput	820.6 TFLOPS/GPU
MFU	41.5%
GPU memory	92.8 GB of 141
Improvement	+10.5% throughput
Step time	9.05 s / iteration

+10.5% from changing ONE parameter
Companion: LoRA finetune PP=4→1 gave +53.7%
The pipeline bubble was a red herring - it was message count

Bottleneck 3 · Communication-bound: NeMo Qwen3 MoE Training (H200/B300)

↓ How It Started

Model	Qwen3-30B-A3B MoE
Config	PP=4, EP=8, VP=4
Throughput	318 TFLOPS
All-to-All	42-51% of GPU time
DeepEP	Force-enabled on EFA
CUDA Graphs	Enabled (default)

⚡ What Changed

DeepEP off	Was -34% on EFA (NVLink-opt)
CUDA graphs off	Deadlock after 3 warmup iters
VP=4→VP=2	Reduced PP comms by 20%
Overlap off	Comp-comm overlap = -20%
B300 memory	268 GB enables larger MBS
Lesson	Disable NVLink features on EFA

↑ How It Ended

Model	Qwen3-235B-A22B MoE
Config	PP=4, EP=8, VP=2
Throughput	240,694 tok/s
All-to-All	Still dominant (expected)
DeepEP	Disabled (NCCL optimal)
CUDA Graphs	Disabled

1.55x more throughput
Don't trust "common knowledge"
Profile your workload every time

Same Hardware, Different Traces: FSDP vs Pipeline Parallel

≡ FSDP (Llama)

Strategy	TP=1, PP=1, FSDP
EFA / node	96.7 GB/s avg
% of 200 GB/s	48% of theoretical
Collective	AllGather + ReduceScatter
Pattern	Large, overlappable transfers
NICs used	many NICs engaged

⚡ What the Trace Shows

Low net %	means OPPOSITE things
FSDP	bandwidth-rich, overlapped
PP+EP	latency-bound, serialized
Same HW	P5en, 16× 400G EFA both
Diagnosis	Read trace SHAPE, not the %
Lesson	Overlap ≠ utilization

➤ PP+EP (Qwen3)

Strategy	PP=4, EP=8 (Qwen 235B)
EFA / node	3.2 GB/s avg
% of 200 GB/s	1.6% of theoretical
Collective	32 MiB P2P SendRecv
Pattern	Small, serial, latency-bound
NICs used	only 1-2 of 16 NICs

FSDP 96.7 GB/s vs PP+EP 3.2 GB/s - same hardware
30x difference in EFA throughput, by strategy alone
The profiler trace shape tells you which regime you are in

Bottleneck 5 · Production VLM (AV perception): Qwen2.5-VL-72B

↓ How It Started

Config	TP=8, PP=4 (default recipe)
Throughput	40,000 tok/s
MFU	~15% (estimated)
GPU memory	69 GB vs 29 GB (imbalanced)
Problem	Vision encoder → PP imbalance
TP	Over-parallelised on 268 GB

⚡ What Changed

TP 8→4	Less over-parallelisation
PP 4→1	Eliminates stage imbalance
MBS 1→2	268 GB enables larger batch
Result	+331% (4.3×) from config only
Root cause	Vision encoder skews PP stages
Lesson	Profile before trusting defaults

↑ How It Ended

Config	TP=4, PP=1, MBS=2
Throughput	172,041 tok/s
MFU	~52.1%
GPU memory	Balanced across GPUs
Improvement	+331% throughput
TFLOPS	~1,173 per GPU

331% more throughput
TPOT* reduced from 3.2ms to 0.74ms
Supports 4.3x more users on the same hardware



The repeatable checklist you leave with

#	Step	What you do	Why
1	Entitlement first	Know the ceiling (% of peak TFLOPS / HBM BW / EFA)	A raw number means nothing without the ceiling
2	Always-on counters	DCGM / sar / EFA / nvidia-smi find the bound subsystem	Low overhead, run on every job
3	nsys for attribution	Confirm the trace signature matches the class	Profile a few steps (~0.6% cost)
4	One lever at a time	Matched batch / concurrency, then re-measure vs ceiling	Isolates cause and effect
5	Distrust inherited configs	PP/EP/recipe tuned for a different GPU count or shape	The #1 silent throughput killer in our campaign

Compute · Memory · Communication · I/O — name it, then fix it.

Call to Action

<https://github.com/awslabs/awsome-distributed-ai>

ML Training Reference Architectures & Tests

This repository contains reference architectures and test cases for distributed model training with [Amazon SageMaker HyperPod](#), [AWS ParallelCluster](#), [AWS Parallel Computing Service \(PCS\)](#), [AWS Batch](#), and [Amazon EKS](#). The test cases cover different types and sizes of models as well as different frameworks and parallel optimizations (PyTorch DDP/FSDP, Megatron-LM, NeMo...).

<https://bit.ly/smhp-ai>

AI on SageMaker HyperPod

Optimized Blueprints for deploying high performance clusters to train, fine tune, and host (inference) models on Amazon SageMaker HyperPod

1,000+
GPUS

40%
FASTER TRAINING

99%
UPTIME

Orchestrated
by EKS



Orchestrated
by SLURM

Thank you!

Paulo Aragao

LinkedIn: paragao

Aravind Neelakantan

LinkedIn: aravind-neelakantan

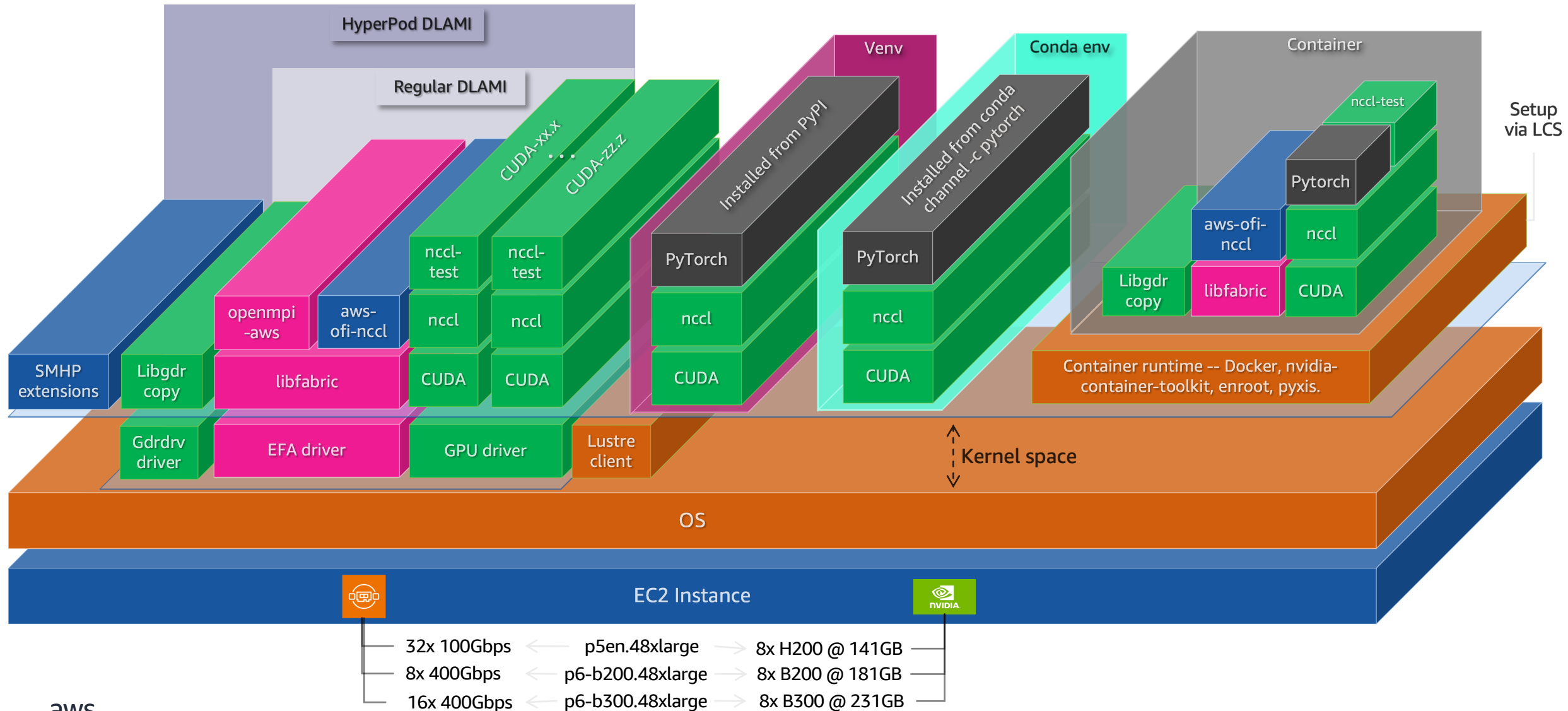
Session Survey



Five Bottlenecks, Five Fixes: Systematic Performance
Diagnosis for PyTorch Distributed Training

Appendix & backup slides

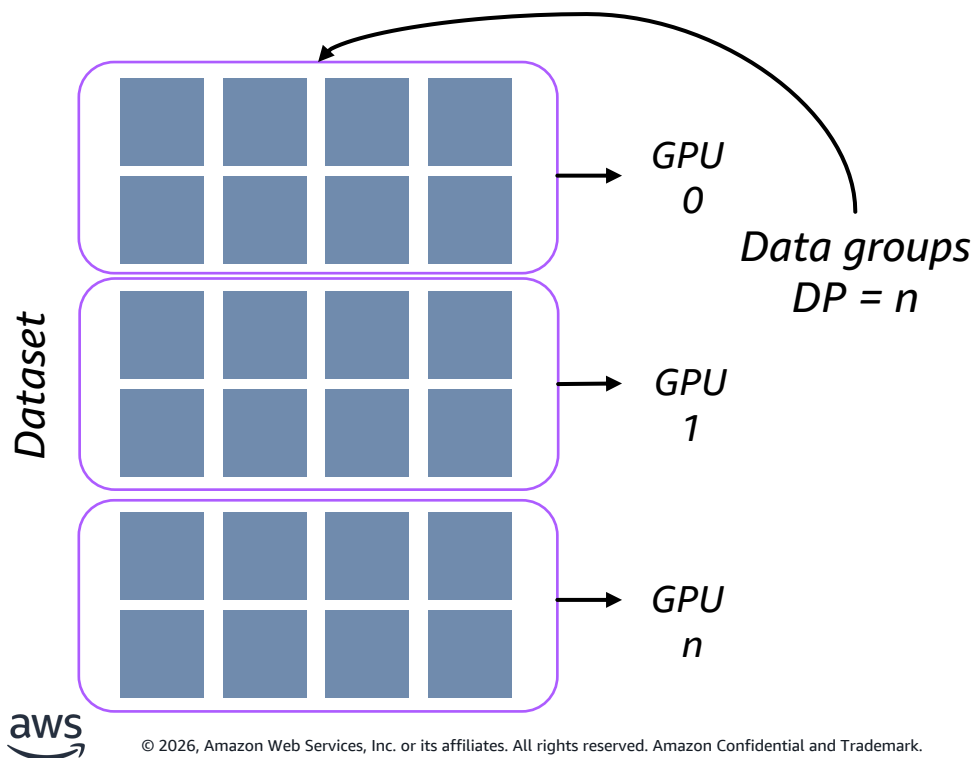
It's not all about hardware.... The software stack



Parallelism strategies help distribute your workload

Data Parallelism

Replicates the model, shard the data across GPU groups



Tensor Parallelism

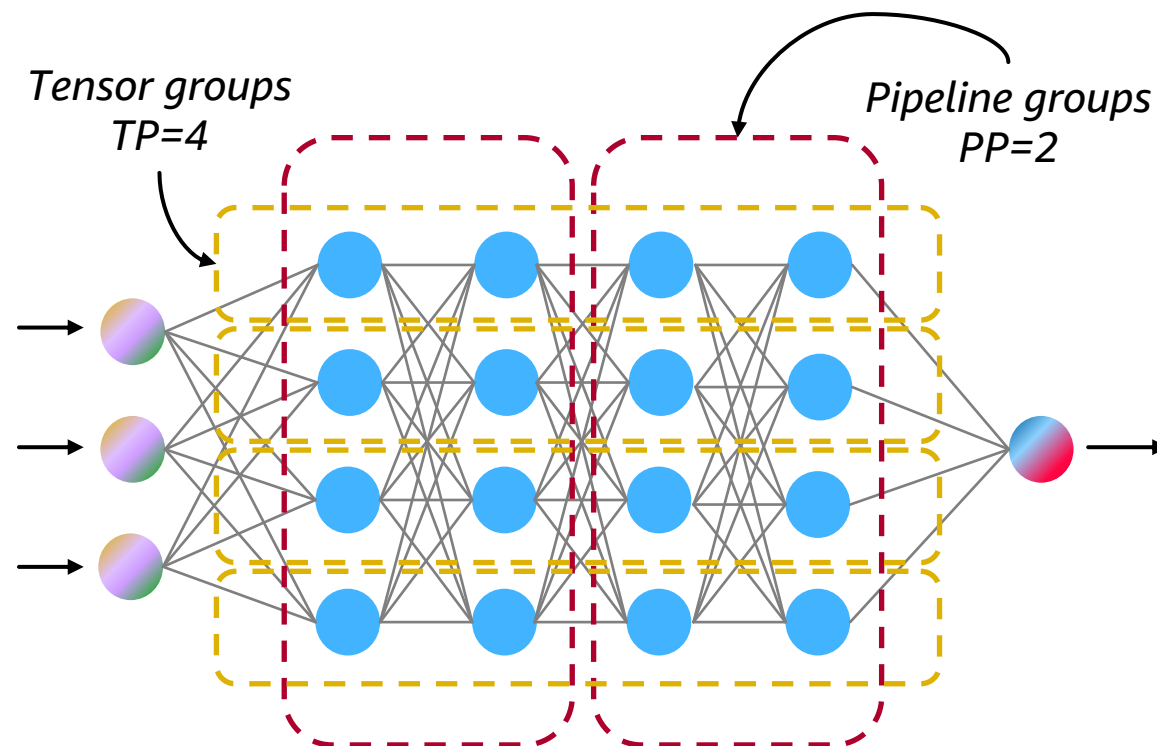
Split the model by distributing layers across GPUs

Pipeline Parallelism

Shard the model in stages and coordinate across GPU groups

Expert Parallelism

Distribute experts across GPUs, routing tokens per expert



Disaggregated Inference

KV per token = $2 \times \text{kv_heads} \times \text{head_dim} \times \text{layers} \times \text{bytes_per_element}$

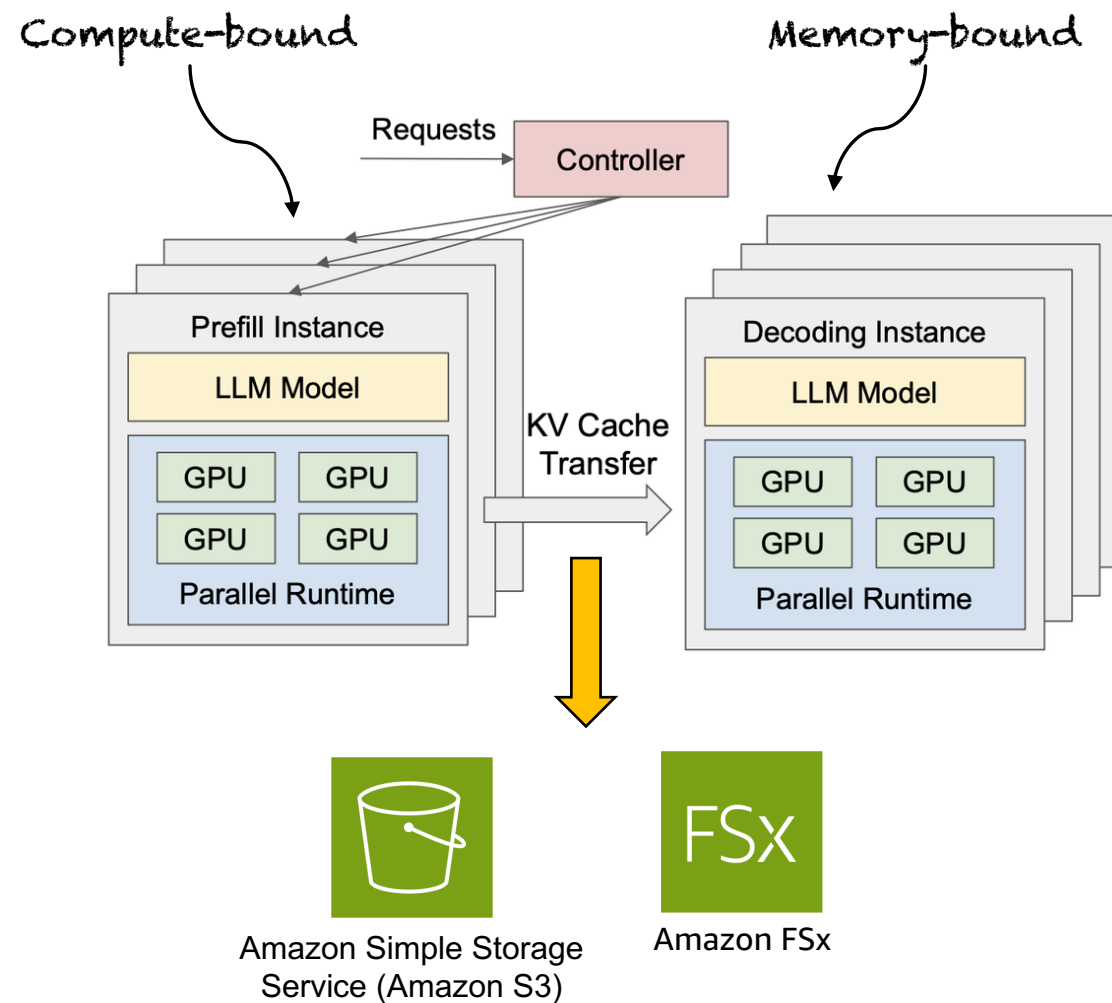
Total KV cache = $\text{seq_length} \times \text{KV_per_token}$

Example (Llama 3.1 70B, seq=4096, BF16):

Per token: $2 \times 8 \text{ kv_heads} \times 128 \text{ dim} \times 80 \text{ layers} \times 2 \text{ bytes} = 327 \text{ KB/token}$

At 4096 tokens: $327 \text{ KB} \times 4096 = 1.3 \text{ GB per request}$

Serving 128 concurrent requests: $128 \times 1.3 \text{ GB} = 166 \text{ GB just for KV cache}$



KV Cache: Infrastructure Impact

What	Where It Comes From	Persists?
Model weights (W_Q , W_K , W_V , W_O)	Pre-training + post-training	✓ Saved to disk
KV cache (K, V tensors)	Generated at inference (prefill)	✗ Ephemeral, per-request
Activations	Computed per forward pass	✗ Discarded after use

Phase	Bottleneck	Optimisation
Prefill	Compute (TFLOPS)	Tensor Parallelism, larger batch
Decode	Memory bandwidth (GB/s)	KV cache quantisation, GQA, PagedAttention
KV cache growth	GPU memory capacity	vLLM PagedAttention, prefix caching, offload

Case Study: RLVR GRPO Qwen3-8B on EKS (32 × H200)

↓ How It Started

Step time	246 seconds
Throughput	21 tok/s
MFU	0.12%
Actor update	216s (88% of step)
Padding waste	69% of actor compute
GSM8K accuracy	7% (untrained)

⚡ What Changed

Dynamic batch	use_dynamic_bsz=True → 3.9×
FSDP2	strategy=fsdp2 → -7% mem
Fwd prefetch	forward_prefetch=True
vLLM KV cache	gpu_mem_util=0.7 → +17% KV
Total speedup	2.9× wall-clock (config only)
Lesson	Padding = invisible waste → profile!

↑ How It Ended

Step time	85 seconds
Throughput	107 tok/s
MFU	0.89%
Actor update	56s
Padding waste	Eliminated
GSM8K accuracy	94.1% (step 195)

5.1x more tokens/second
75% faster actor updates
Achieved 94.1% accuracy on GSM8K



Understand your model (theoretical overview)

<https://github.com/paragao/ai-model-calculator>

AI Model Calculator estimates per-GPU memory, batch configuration, wall-clock time and communication overhead for an LLM training run on AWS GPU instances, before you reserve them. Dense and MoE models; Python standard library only.¹

Mode	Question it answers	You provide
A	How many instances to finish by date X?	model, dataset tokens, target time
B	How long will training take?	model, dataset tokens, instance count
C	What is the minimum hardware to fit it?	model, instance type

Six phases

- 1 Memory / GPU** weights, grads, optimizer, activations, logits, NCCL + MoE buffers, fragmentation
- 2 Batch** micro-batch × gradient-accumulation grid that fits
- 3 Time** $(6 \cdot N_{\text{active}} \cdot D + \text{attention}) \div (\text{GPUs} \times \text{peak} \times \text{MFU}) + \text{PP bubble} + \text{exposed comm}$
- 4 ZeRO comm** reduce-scatter / all-gather volume per stage
- 5 MoE all-to-all** dispatch + combine per MoE layer
- 6 PP send/recv** NVLink (intra-node) vs EFA (inter-node), bubble %

Inputs built-in catalog (Llama 3.x/4, Qwen 2.5/3/3.5, DeepSeek V3/V4) or custom JSON · hardware presets p5, p5en, p6-b200, p6-b300, g5, g6, g6e or custom JSON · TP / PP / CP / EP / VP, sequence length, precision, MFU (default 0.40)¹

Runs as CLI · Docker · Kiro Power · Amazon Quick skill · Claude Code skill¹

Predicted vs measured: Llama-3.1-70B BF16, TP2/PP4/CP2, 64 × H200: 108.08 GB reserved, ZeRO-2 prediction $1.01 \times^2$

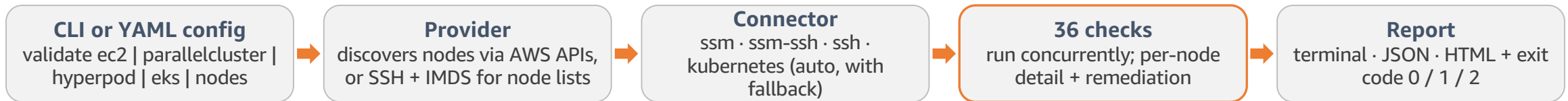
Read the output correctly: MFU is an **input**, not a prediction; time scales linearly with it.² FP8 changes the compute peak, not memory: a controlled 70B LoRA pair used +8.7% *more* memory in FP8.²

[1] github.com/paragao/ai-model-calculator README [2] github.com/paragao/ai-model-calc-skill SKILL.md (Phase 1 & 3 notes; validation against dqxc job 179) [3] [Measured 108.08 GB: portal pretrain-aws/70b-bf16/p5en-8nodes_job_179](#)



Validate the cluster before you benchmark

AI/ML Cluster Validator is an open-source CLI that checks networking, EFA, GPU, SSH, Slurm, software and, optionally, performance across EC2, ParallelCluster, SageMaker HyperPod, EKS or a plain node list.¹



Category	#	What it validates
networking	5	Subnets, private IPs, placement groups, capacity reservations, security groups
efa	5	Interface attachment, kernel module, devices, libfabric, OFI-NCCL plugin
gpu	10	nvidia-smi, GPU count, CUDA, Fabric Manager, NVLink, persistence, GDRCopy, memory, thermal
ssh	2	Passwordless SSH, SSH config
slurm	7	Controller, node health, job submit, GPU scheduling, Pyxis, Enroot, MPI
software	1	Version inventory + GPU-family compatibility matrix
eks	5	Device plugins, allocatable resources, GPU/EFA validation pods
performance	1	NCCL, nvbandwidth, STREAM, EFA, storage I/O (opt-in, 2–5 min)

Run it

HyperPod, GPU + EFA only

```
cluster-validator validate hyperpod --cluster-name <name> -  
-checks gpu,efa
```

EKS (creates validation pods)

```
cluster-validator validate eks --cluster-name <name>
```

Slurm nodes from the head node, no AWS APIs

```
cluster-validator validate nodes --nodes n1,n2 --scheduler  
slurm
```

Add performance benchmarks

```
... -p --perf-image /fsx/containers/perf-validator.sqsh
```

CI gate

```
cluster-validator -n validate nodes --nodes n1,n2 --output  
json --exit 0 = passed (warnings OK) · 1 = failure · 2 = runner
```

Gate every benchmark on a clean run (exit code 0); a broken EFA, NVLink or Fabric Manager check invalidates the numbers

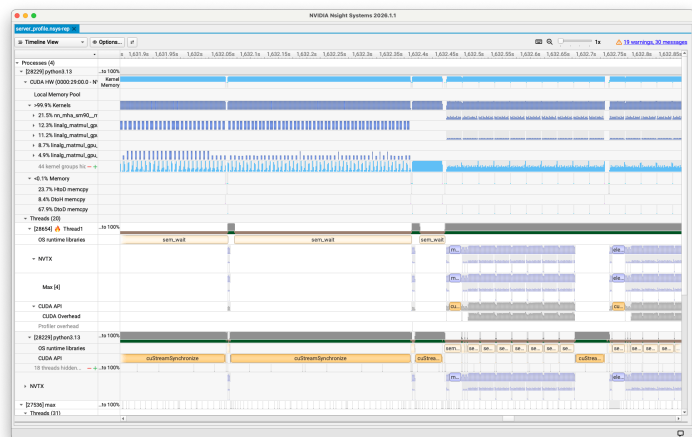
Profiling and performance analysis tooling

NVIDIA nsys – NSight System

When: you need attribution, i.e. where one step's time goes across CPU threads, CUDA API, kernels, memcopy, NCCL and NVTX ranges.

Run: `nsys profile -t cuda,nvtx,osrt,nccl --gpu-metrics-devices=all --capture-range=cudaProfilerApi -o step <cmd>`, then `nsys stats` for summaries. GPU metrics add SM / Tensor Core activity and PCIe, NVLink and HBM throughput.¹

Cost: profile a few steps. Llama-3.1-70B, 64 × H200: 526.39 TFLOPS/GPU with nsys vs 529.33 without (-0.6%).²



Reading this trace

Kernels fill >99.9% of the GPU row; host threads wait in `sem_wait` / `cuStreamSynchronize`. The GPU is the busy side: look at kernel efficiency next, not the CPU or data loader.

System Stats

CPU `sar -u ALL`, `sar -P ALL`: per-core saturation (data-loader workers), %iowait, %steal

RAM & swap `vmstat 1` (si/so, r, b); `sar -W` swap pages/s; `sar -B` majflt/s, pgscand/s (direct reclaim = stalled process)

Disk `sar -d`: await, aqu-sz. %util is not a limit for NVMe or RAID, which serve requests in parallel

Host NIC `sar -n DEV,EDEV`: rxkB/s, txkB/s, %ifutil, drops

Record the whole run with `sar -o file 1` in the background; replay with `sar -f`.³

EFA Top

Counters `rdma -p statistic show or /sys/class/infiniband/<dev>/ports/1/hw_counters/`: cumulative since launch

Throughput $\Delta(\text{tx_bytes}, \text{rx_bytes}, \text{rdma_read_bytes}, \text{rdma_write_bytes}) \div \Delta t$, compared with 400 GB/s per p5/p5en instance

Health `rx_drops`, `retrans_bytes / pkts`, `retrans_timeout_events`, `impaired_remote_conn_events`, `unresponsive_remote_events` (Nitro v4+)

Fleet CloudWatch agent EFA metrics; Container Insights on EKS (no retrans / impaired counters)⁴

