APPENDIX C · BACK MATTER

# Glossary of Key Terms

*Production-precision definitions for every term a Principal Performance Architect must know — organized alphabetically with chapter cross-references*

**Coverage:** 65+ terms, A–Z

**Scope:** GPU architecture · Training · Inference · Networking · Cluster ops

**Cross-refs:** Chapters 1–18

> *"A precise vocabulary is not pedantry — it is the prerequisite for precise reasoning. An engineer who confuses 'bandwidth-bound' with 'memory-bound' cannot correctly diagnose a production performance regression, no matter how talented they are."*

**HOW TO USE THIS GLOSSARY**

Terms are defined at the precision required for production use, not introductory accuracy. Cross-reference tags (→ Ch N) point to the chapter where the concept is developed in full with worked examples and profiling recipes. Terms in *italic style* within a definition are themselves defined elsewhere in this glossary. For interview prep, prioritize terms marked with chapter references spanning multiple chapters — those concepts appear at the architectural intersection of the book.

## A

5 TERMS

### AllGather

A collective communication operation where each process contributes a shard, and after the operation every process holds all shards concatenated. Used in **ZeRO**/**FSDP** to reconstitute full parameter tensors before the forward pass. Opposite of **ReduceScatter**. In ZeRO-3, AllGather runs before each forward and backward layer, and the gathered parameters are freed immediately after to maintain the sharded memory footprint.

→ Ch 10 Training · → Ch 14 Networking

### AllReduce

A collective communication operation that reduces (e.g., sums) tensors across all processes and gives every process the result. The backbone of **Data Parallelism** gradient synchronization. Implemented in **NCCL** using Ring or Tree algorithms, or via **NVLS**/**SHARP** for in-network aggregation. Communication volume: 2 × (N-1)/N × message_size per rank (Ring). On H100 NVLink clusters, NVLS reduces this to a single pass through NVSwitch hardware, achieving ~2× lower latency than Ring for large messages.

→ Ch 10 Training → Ch 14 Networking

---

### Arithmetic Intensity AI

FLOPs performed per byte of memory traffic. `AI = FLOPs / bytes_transferred`. The key axis in the **Roofline** model. Operations with AI below the **ridge point** are memory-bandwidth-bound; above are compute-bound. Example: a decode GEMV at batch=1 has AI ≈ 1 FLOP/byte — deep in memory-bound territory on any GPU. A training GEMM at M=N=K=4096 has AI ≈ 1,365 FLOP/byte — solidly compute-bound on H100.

→ Ch 1 Mindset → Ch 7 Kernels → Ch 12 Benchmarking

---

### Autoregressive Decoding

The standard token generation algorithm: generate one token at a time, each conditioned on all prior tokens. Requires loading all model weights each step — inherently **memory-bandwidth-bound** at small batch sizes. The fundamental reason decode throughput is limited by HBM bandwidth, not compute. For a 70B BF16 model (140 GB) on H100 (3.35 TB/s), minimum token time = 140 GB / 3,350 GB/s ≈ 42ms, regardless of sequence length.

→ Ch 6 Inference → Ch 11 KV Cache

---

### AWQ Activation-Aware Weight Quantization

A **PTQ** method that identifies "salient" weight channels — those corresponding to large-magnitude activations — and scales them before quantization to protect them from quantization error. Typically outperforms **GPTQ** for knowledge-intensive tasks. Uses per-group-128 granularity. Does not require Hessian computation, making it faster to apply than GPTQ on large models. Widely deployed for INT4 weight-only quantization of 7B–70B models.

→ Ch 8 Quantization

## B

3 TERMS

### BF16 BrainFloat16

A 16-bit floating point format with 8 exponent bits and 7 mantissa bits. Same dynamic range as FP32 (±3.4×10³⁸) but lower precision per bit. The standard training format for LLMs since GPT-3. Replaced **FP16** for training because its wider exponent range eliminates gradient overflow — the primary failure mode of FP16 mixed-precision training at scale. H100 SXM: 989 TFLOPS BF16 Tensor Core throughput. All serious LLM training since 2021 uses BF16 as the primary compute dtype.

→ Ch 8 Quantization → Ch 3 GPU Architecture

---

### Blocked Roofline

See **Roofline Model**. A variant that accounts for cache hierarchy: instead of using peak HBM bandwidth as the roof slope, it uses the effective bandwidth achieved after accounting for L2 and SRAM reuse. FlashAttention's performance advantage is best explained by the blocked roofline — its SRAM tiling dramatically increases the effective arithmetic intensity by eliminating HBM trips for intermediate attention scores.

→ Ch 7 Kernels → Ch 4 Memory

---

### Bubble Fraction

The fraction of pipeline parallel compute time wasted on idle pipeline stages. Formula: `(PP-1) / (PP-1+M)` where PP = pipeline stages and M = microbatches. Target: <10%. At PP=8 and M=16: bubble = 7/23 = 30.4% — unacceptably high. At M=64: bubble = 7/71 = 9.9% — acceptable. The fundamental reason pipeline parallelism requires large batch sizes to achieve reasonable efficiency.

→ Ch 10 Training

## C

5 TERMS

---

### Chunked Prefill

An inference optimization that splits long prompts into fixed-size chunks and interleaves prefill and decode steps within a single scheduler iteration. Reduces **TTFT** variance by sharing GPU time between prefill and decode rather than blocking all decode requests during long prefill. Chunk sizes of 512–2048 tokens are typical. When a 64K-token prompt would otherwise monopolize the GPU for 8+ seconds, chunking it into 512-token pieces allows ongoing decode requests to make forward progress, keeping P99 TTFT within SLA.

→ Ch 6 Inference → Ch 11 KV Cache

---

### Collective Operation

A communication primitive that involves all processes in a group simultaneously: **AllReduce**, **AllGather**, **ReduceScatter**, Broadcast, All-to-All. Contrast with point-to-point operations (send/recv). Collectives are the dominant communication pattern in distributed training and must be overlapped with compute to avoid degrading GPU utilization. The choice of collective algorithm (Ring vs Tree vs NVLS) significantly affects performance depending on message size and topology.

→ Ch 14 Networking → Ch 10 Training

---

### Compute-Bound

A workload regime where the GPU's arithmetic units (Tensor Cores) are the bottleneck. Characterized by arithmetic intensity above the **roofline ridge point**. Optimization targets: increase precision utilization (**FP8**), improve tile alignment (multiples of 16/128), reduce kernel launch overhead, maximize warp occupancy. Compute-bound operations include: large GEMM (training, large-batch prefill), dense convolutions. Cannot be accelerated by faster HBM — the bottleneck is Tensor Core throughput.

→ Ch 1 Mindset → Ch 7 Kernels

---

### Continuous Batching

An inference scheduling strategy that adds new requests to the active batch as soon as existing requests finish individual decode steps, rather than waiting for the entire batch to complete. Achieves near-100% GPU utilization without increasing **TTFT**. Industry standard in **vLLM**, TRT-LLM, SGLang. Before continuous batching, servers had to wait for the slowest request in a batch to finish — a sequence generating 1000 tokens would block all others until complete. With continuous batching, that slot is reclaimed token-by-token.

→ Ch 6 Inference

---

### Context Parallelism CP

A parallelism strategy that splits the sequence dimension across GPUs for long-context training. Each GPU holds S/CP query positions and attends over the full key-value set via All-to-All communication. Activates at S > 32K tokens where sequence-length-proportional activation memory exceeds single-GPU HBM capacity. CP=4 reduces per-GPU activation memory from S×d to S/4×d at the cost of an All-to-All per attention layer. Used at Meta for Llama-3 128K context training.

→ Ch 10 Training → Ch 14 Networking

## D

4 TERMS

### Data Parallelism DP

A distributed training strategy where each worker holds a full model replica and processes different data shards. Gradients are synchronized across workers via **AllReduce** after each backward pass. Simplest parallelism form — no model changes required. Scales linearly until the AllReduce communication time approaches the compute time. Pure DP is not viable for models larger than a single GPU's memory; see **FSDP**/**ZeRO** for sharded variants.

→ Ch 10 Training

---

### DCGM Data Center GPU Manager

NVIDIA's hardware telemetry daemon. Reads GPU silicon counters (SM utilization, temperature, ECC errors, power, NVLink bandwidth) at 1-second resolution. Exposes metrics to **Prometheus** via `dcgm-exporter`. The authoritative source for GPU health data in production clusters. Key metrics: `DCGM_FI_DEV_GPU_UTIL` (SM utilization), `DCGM_FI_DEV_MEM_COPY_UTIL` (memory bandwidth utilization), `DCGM_FI_DEV_ECC_SBE_VOL_TOTAL` (soft ECC errors — leading indicator of GPU failure).

→ Ch 17 Telemetry → Ch 12 Benchmarking

---

### Decode

The autoregressive token generation phase of inference. Memory-bandwidth-bound at small batch sizes because the full model must be loaded from **HBM** for each generated token. Distinguished from **Prefill**. Decode throughput is limited by HBM bandwidth: `tokens/sec = HBM_bandwidth / model_bytes_per_token`. For 70B BF16 on H100: 3.35 TB/s / 140 GB ≈ 23.9 tokens/sec per GPU at batch=1. Increasing batch size improves arithmetic intensity until the GEMM becomes compute-bound, typically at batch 32–128 depending on model size.

→ Ch 6 Inference → Ch 11 KV Cache

---

### DPO Direct Preference Optimization

A post-training alignment method that trains the policy model using a pair of chosen/rejected responses, without requiring a separate reward model. Simpler infrastructure than **PPO**: requires only policy + reference model (2× model memory vs PPO's 3–4×). Mathematically equivalent to optimizing a reward signal implicit in the preference data. Widely used for instruction-following alignment. The primary alternative to PPO for teams without large RL infrastructure budgets.

→ Ch 10 Training

## E

2 TERMS

### ECC Error Correction Code

A hardware mechanism in GPU **HBM** that detects and corrects memory bit errors. Single-bit errors (SBE / correctable) are fixed silently but logged via **DCGM** — high SBE rates are a leading indicator of HBM failure. Double-bit errors (DBE / uncorrectable) cause a **XID** error and require immediate GPU replacement. Enabling ECC reduces effective HBM bandwidth by ~6% on H100 due to additional data encoding overhead. Production clusters always enable ECC despite the bandwidth cost — the alternative is silent data corruption.

→ Ch 17 Telemetry → Ch 3 GPU Architecture

### Expert Parallelism EP

A parallelism strategy for **MoE** models that distributes different experts across GPUs. Each GPU hosts a subset of experts and processes only tokens routed to those experts. Requires All-to-All communication to route tokens to the correct expert GPU. Typically deployed within the **NVLink** domain (EP ≤ 8) to minimize latency — InfiniBand All-to-All latency is prohibitive for per-layer expert routing. DeepSeek-V3 uses EP=64 within a single node via NVLink-C2C, achieving sub-µs expert routing.

→ Ch 13 MoE → Ch 14 Networking

## F

6 TERMS

### Fair-Share Scheduling

A **SLURM** scheduling policy where job priority decreases as a user's historical resource consumption exceeds their allocated share. The fair-share score approaches 1.0 for under-utilizing users (high priority) and 0.0 for over-utilizing users (low priority). Prevents any team from monopolizing the cluster over time. Configured via `PriorityWeightFairshare` in SLURM. Essential for large multi-tenant training clusters shared across research teams with competing priorities.

→ Ch 16 Cluster Mgmt

### FLOP Floating Point Operation

One multiply or add operation. A multiply-accumulate (MAC) counts as 2 FLOPs. Used to measure compute requirements of neural network forward passes. Key formula: Total training FLOPs ≈ 6 × N_params × N_tokens (the "6" accounts for forward pass, backward pass, and gradient accumulation). For Llama-3 70B trained on 15T tokens: 6 × 70B × 15T = $6.3 \times 10^{24}$ FLOPs. At 989 TFLOPS per H100 and 40% MFU: ~50,000 H100-hours required.

→ Ch 1 Mindset → Ch 10 Training

### FlashAttention

A fused attention kernel (Dao et al. 2022) that implements attention in tiles, keeping intermediate K,V tiles in on-chip SRAM rather than materializing the full $O(S^2)$ attention score matrix in HBM. Reduces attention HBM traffic from $O(S^2)$ to $O(S)$. At S=8192 tokens, standard attention materializes a 10.7 GB score matrix per layer — FlashAttention eliminates this entirely. v3 (H100) achieves 585 TFLOPS — 58% of H100 theoretical peak — by exploiting Hopper's asynchronous WGMMA instructions and TMA memory access engine.

→ Ch 4 Memory → Ch 7 Kernels → Ch 11 KV Cache

---

### FP8

An 8-bit floating point format with two variants: E4M3 (4 exponent, 3 mantissa bits, range ±448, used for weights and activations) and E5M2 (5 exponent, 2 mantissa bits, range ±57K, used for gradients). Supported natively on H100/H200/B200 via **Transformer Engine**. Delivers ~2× throughput vs BF16 for GEMM: H100 FP8 peak = 1,978 TFLOPS vs 989 TFLOPS BF16. Requires careful scaling to prevent overflow/underflow — TE manages this automatically via per-tensor `amax_history` tracking.

→ Ch 8 Quantization → Ch 3 GPU Architecture → Ch 10 Training

---

### FSDP Fully Sharded Data Parallel

PyTorch's native implementation of **ZeRO Stage 3**. Shards model parameters, gradients, and optimizer state across all DP workers. Equivalent to `ShardingStrategy.FULL_SHARD`. Requires 3× AllReduce bandwidth vs naive DP but enables training models of arbitrary size — the parameter shard per GPU shrinks linearly with DP degree. At DP=8: each GPU holds 1/8th of parameters at rest, AllGather-ing each layer's weights during computation and discarding them immediately after.

→ Ch 10 Training → Ch 14 Networking

---

### Fusion / Operator Fusion

Merging multiple GPU operations into a single kernel to eliminate intermediate **HBM** round-trips. Two types: Vertical fusion (chain of sequential ops, e.g., RMSNorm + GEMM → one kernel, 2× fewer HBM reads); Horizontal fusion (parallel ops, e.g., Q+K+V projections → single QKV GEMM, 1 launch vs 3). Key optimization in **TorchInductor**, **TensorRT**, **NVFuser**. A fused RMSNorm+GEMM saves ~1.2 GB/layer of HBM traffic on a 70B model — recovering 15–20% of HBM bandwidth otherwise spent on normalization pass.

→ Ch 7 Kernels → Ch 9 Compilers

## G

6 TERMS

### GEMM General Matrix Multiplication

The core compute primitive of neural network layers. C = A × B, where A ∈ ℝ^{M×K}, B ∈ ℝ^{K×N}, C ∈ ℝ^{M×N}. FLOPs = 2×M×K×N. The dominant operation in **prefill** and training — 80–95% of total FLOPs. Compute-bound at large batch sizes (M large); memory-bandwidth-bound at small batch sizes (**decode**). H100 peak: 989 TFLOPS BF16, 1,978 TFLOPS FP8. Dimension alignment to multiples of 128 (BF16) or 64 (FP8) is required for peak Tensor Core utilization.

→ Ch 7 Kernels → Ch 6 Inference → Ch 10 Training

---

### GPUDirect RDMA

An NVIDIA technology that allows the NIC to read from and write to GPU **HBM** directly via PCIe P2P, bypassing CPU stage buffers. Reduces communication latency by ~2 µs per message and frees ~12 CPU cores per DGX node that would otherwise serve as data proxy threads. Required for high-throughput inference disaggregation (prefill-decode disaggregation) where KV cache transfers between nodes must complete in <1ms to maintain SLAs. Works with both InfiniBand and RoCEv2 NICs.

→ Ch 14 Networking

---

### GQA Grouped Query Attention

An attention variant where multiple query heads share a single key-value head group. Reduces **KV cache** size by H_q/H_kv (e.g., 8× reduction for GQA-8 with 64 query heads). Standard in LLaMA-3, Mistral, Qwen. Enables much higher concurrent serving capacity vs **MHA** at minimal quality cost. Memory formula: KV_cache_bytes = 2 × L × H_kv × D_head × S × dtype_bytes. For Llama-3 70B (GQA-8, L=80, H_kv=8, D_head=128): 2×80×8×128×S×2 = 327,680×S bytes per sequence.

→ Ch 2 Transformer → Ch 11 KV Cache

---

### GPTQ

A **PTQ** method that applies second-order (Hessian-based) error correction during weight quantization. Processes weight matrices column by column, propagating quantization error to remaining columns to minimize output error. Achieves approximately -1.3% MMLU delta for INT4 quantization of LLaMA-3 70B — the current state of the art for static quantization accuracy. Slower to apply than **AWQ** (requires Hessian computation) but often superior on reasoning tasks. The default quantization baseline for production INT4 deployments.

→ Ch 8 Quantization

---

### Gradient Clipping

Rescaling the gradient vector when its L2 norm exceeds a threshold (typically 1.0). Prevents gradient explosion during training — a divergent backward pass that would corrupt model weights. Applied before the optimizer step: `torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)`. A clipping event is a diagnostic signal: if clipping fires on >5% of steps, the learning rate may be too high or the model architecture has numerical instabilities. Log clip fraction as a training health metric.

→ Ch 10 Training

---

### GRES Generic Resources

**SLURM**'s mechanism for tracking non-CPU/memory resources like GPUs. Specified in job scripts with `--gres=gpu:h100:8` to request 8 H100 GPUs. Configured per-node in `gres.conf` with explicit GPU topology and binding. Used with `--gpu-bind=closest` to ensure NIC-GPU affinity for optimal **RDMA** throughput. Without proper GRES configuration, multi-tenant GPU clusters allocate GPUs without NIC affinity awareness — degrading collective communication performance by 30–50%.

→ Ch 16 Cluster Mgmt

## H

2 TERMS

---

### HBM High-Bandwidth Memory

3D-stacked DRAM attached directly to the GPU die via a wide silicon interposer bus (1024–2048 bit). Provides dramatically higher bandwidth than GDDR at the cost of lower capacity per module and higher cost per GB. The most critical performance resource for memory-bandwidth-bound AI workloads. Current standards: HBM3e (H200, B200) at 4.8–8.0 TB/s. HBM2e (A100): 2.0 TB/s. HBM3 (H100 SXM): 3.35 TB/s. The bandwidth difference between A100 and H100 (2.0 vs 3.35 TB/s) translates directly to 1.68× faster decode throughput for any memory-bound workload.

→ Ch 3 GPU Architecture → Ch 4 Memory

---

### HFU Hardware FLOP Utilization

Like **MFU** but accounts for recomputed activations. `HFU = (forward_FLOPs + recompute_FLOPs) / (peak_FLOPs × time)`. HFU ≥ MFU always. The difference between HFU and MFU reveals the overhead of activation checkpointing: if HFU = 55% and MFU = 40%, the recompute overhead is (55-40)/55 = 27% of actual compute — an unusually high recompute fraction that suggests the checkpointing granularity should be reduced. Use both metrics together when analyzing training efficiency.

→ Ch 10 Training → Ch 12 Benchmarking

# I

1 TERM

### Inference

Running a trained model to generate outputs. Divided into two phases: **Prefill** (process the input prompt, compute and cache KV projections for all tokens simultaneously — compute-bound) and **Decode** (generate output tokens autoregressively, one token per step — memory-bandwidth-bound). The distinction matters architecturally: prefill and decode have fundamentally different optimization targets, hardware efficiency characteristics, and SLA metrics (TTFT vs TPOT). Disaggregated inference systems assign them to separate hardware pools to optimize each independently.

→ Ch 6 Inference → Ch 11 KV Cache

# K

2 TERMS

### KV Cache Key-Value Cache

The cached tensor of Key and Value projections for all prior tokens in an autoregressive sequence. Stored in **HBM**. Enables O(N) instead of O(N²) decode compute per step by reusing previously computed KV projections. The single most critical memory resource in inference systems — KV cache capacity directly determines how many concurrent sequences the system can serve. KV memory per token: `2 × L × (H_kv/H_q) × D_head × dtype_bytes`. At Llama-3 70B BF16: ~327KB per token. At 128K context: 41.9 GB of KV per sequence — exceeding the H100's 80 GB capacity with only 2 sequences.

→ Ch 11 KV Cache → Ch 6 Inference

### KV Cache Cascade

A failure mode where high **KV cache** utilization (>~88%) causes each new request to evict an existing sequence, which triggers eviction of another, exponentially growing the overhead. P99 **TTFT** can spike 10–50× in seconds — a production incident pattern. Prevention: implement admission control that stops accepting new requests at 82% KV utilization, not 95% or 100%. Monitor via `vllm:gpu_cache_usage_perc` Prometheus metric with alerting at 80% to provide 10 minutes of warning before cascade onset.

→ Ch 11 KV Cache → Ch 17 Telemetry

# L

2 TERMS

### LoRA Low-Rank Adaptation

A parameter-efficient fine-tuning method that adds trainable low-rank matrices (A ∈ ℝ^{d×r}, B ∈ ℝ^{r×d}) alongside frozen base model weights. Only A and B are updated during fine-tuning. Memory cost: proportional to r × d per layer, not d × d. For Llama-3 70B with r=16, fine-tuning adds ~500M trainable parameters vs 70B for full fine-tune — a 140× memory reduction. During inference, LoRA adapters can be merged: W_merged = W_base + (α/r) × BA, with zero inference overhead. Multiple adapters can be hot-swapped for multi-task serving.

→ Ch 10 Training → Ch 8 Quantization

### Lustre

A widely-deployed parallel distributed filesystem for HPC and AI training clusters. Decouples metadata (MDT — Metadata Target) from data (OST — Object Storage Target), enabling thousands of clients to access the same filesystem concurrently. Key optimization: `lfs setstripe -c -1 /path/to/checkpoint/` to stripe checkpoint files across all OSTs for maximum write bandwidth — this must be set before data is written; it cannot be changed retroactively. Production AI clusters typically achieve 200–500 GB/s aggregate Lustre bandwidth for checkpoint I/O, vs 3–4 TB/s for HBM — making I/O the bottleneck for frequent checkpointing.

→ Ch 15 Storage I/O

## M

7 TERMS

### MFU Model FLOP Utilization

The fraction of theoretical GPU FLOPS actually used by the model. `MFU = (6 × N_params × tokens_per_sec) / (N_GPUs × peak_FLOPS)`. The primary efficiency metric for distributed training. Target: 45–60% on H100 clusters. Below 40%: significant bottleneck present — diagnose with pipeline bubble analysis, communication profiling, and data loading checks. Above 60% sustained: excellent system-level optimization. MFU of 50% on a 1,024 H100 cluster = 506 PFLOPS effective — 50× the compute of a single H100.

→ Ch 10 Training → Ch 12 Benchmarking

### MHA Multi-Head Attention

Standard attention with H_q = H_kv query/key/value heads. Maximum quality, maximum **KV cache** memory footprint. The original attention formulation from Vaswani et al. (2017). For a model with 64 attention heads at D_head=128, each token requires 2 × L × 64 × 128 = 16,384×L bytes of KV cache in BF16. Largely replaced by **GQA** in modern LLMs for serving efficiency — GPT-4 is one of the last large production models to use full MHA at scale.

→ Ch 2 Transformer → Ch 11 KV Cache

---

### MIG Multi-Instance GPU

NVIDIA H100+ feature that partitions a single GPU into up to 7 isolated instances, each with dedicated SM slices, L2 cache partitions, and HBM bandwidth. Enables safe multi-tenant GPU sharing for inference of smaller models (7B–13B) that don't saturate a full H100. Each MIG instance has hardware-enforced memory protection — one tenant's workload cannot read or corrupt another's HBM. MIG instance sizes: 1g.10gb (10GB, 1/7 GPU) to 7g.80gb (80GB, full GPU). Primary use case: serving many small models on expensive GPUs to maximize utilization and amortize cost.

→ Ch 3 GPU Architecture → Ch 6 Inference

---

### MLA Multi-Head Latent Attention

DeepSeek's architecture innovation that caches a compressed low-rank latent vector (d_c = 512) instead of full K,V heads. Reconstructs K,V at attention time from the latent via learned up-projection matrices. Achieves ~57× **KV cache** memory reduction vs standard MHA with near-zero quality loss. The KV cache per token: 2 × L × 512 bytes vs 2 × L × H × D_head bytes for MHA. Enables 128K+ context serving at a fraction of the memory cost. Standard in DeepSeek-V2/V3/R1.

→ Ch 2 Transformer → Ch 11 KV Cache → Ch 13 MoE

---

### MoE Mixture of Experts

An architecture that replaces each dense FFN layer with N expert FFNs, routing each token to K active experts (K << N). Compute ∝ active parameters; quality ∝ total parameters — the key insight that enables scaling model capacity without proportionally scaling compute. DeepSeek-V3: 671B total parameters, 37B active per token at K=8, N=256. Training FLOPs match a 37B dense model while achieving quality superior to many 70B dense models. Requires **Expert Parallelism** and All-to-All communication for distributed deployment.

→ Ch 13 MoE → Ch 14 Networking

---

### MQA Multi-Query Attention

Extreme version of **GQA** with H_kv = 1: all query heads share a single K,V head. Maximum KV compression (~H_q× reduction vs **MHA**) but noticeable quality degradation on complex reasoning tasks. Used in early Falcon and PaLM models. Largely superseded by **GQA** which achieves most of the memory savings with minimal quality loss via 4–8 KV groups rather than a single group. Still used in specialized deployments where KV memory is the hard constraint and quality can be sacrificed.

→ Ch 2 Transformer → Ch 11 KV Cache

## N

5 TERMS

### NCCL NVIDIA Collective Communications Library

NVIDIA's library for GPU collective operations. Implements **AllReduce**, **AllGather**, **ReduceScatter**, All-to-All. Automatically selects Ring, Tree, or NVLS algorithms based on topology and message size. Critical NCCL environment variables: `NCCL_DEBUG=INFO` for logging, `NCCL_IB_DISABLE=0` to enable InfiniBand, `NCCL_NET_GDR_LEVEL=5` for GPUDirect. NCCL 2.21+ includes automatic topology detection; older versions require explicit topology files for optimal rail-optimized routing in fat-tree fabrics.

→ Ch 14 Networking → Ch 10 Training

### NF4 NormalFloat 4-bit

A 4-bit quantization format (Dettmers et al.) optimized for normally-distributed weights. Uses a non-uniform quantization grid with more levels near zero, matching the approximate normal distribution of transformer weights. Achieves lower quantization error than INT4 for the same bit width. Used in **QLoRA**: the base model is stored in NF4 (1/4 original size) while LoRA adapters are trained in BF16. A 70B model in NF4 requires ~35 GB — fitting in a single H100 — enabling fine-tuning that would otherwise require multi-node infrastructure.

→ Ch 8 Quantization

### NVFP4

NVIDIA's 4-bit floating-point format for Blackwell (B200). Format: 1-bit sign + 2-bit exponent + 1-bit mantissa with per-group (16 elements) FP8 scale factor. Peak throughput: ~18 PFLOPS on B200, vs 9 PFLOPS BF16 — approximately 2× vs FP8. Enables ~8× throughput vs BF16 for weight-only inference. The per-group scaling maintains sufficient accuracy for serving (vs the potential overflow of naive 4-bit float formats). Production deployments report <1% quality degradation on MMLU vs BF16 baseline with NVFP4 activation quantization.

→ Ch 3 GPU Architecture → Ch 8 Quantization

### NVLink

NVIDIA's high-bandwidth GPU-to-GPU interconnect implemented via NVSwitch chips on DGX nodes. Provides non-blocking all-to-all connectivity within an 8-GPU node without PCIe bottleneck. Bandwidth: 900 GB/s bidirectional per GPU (NVLink 4.0, H100), 1.8 TB/s (NVLink 5.0, B200). The reason **Tensor Parallelism** must be confined within a node (TP ≤ 8): NVLink all-reduce takes ~1µs vs IB all-reduce at ~5–10µs — the 5–10× latency difference makes cross-node TP impractical for per-layer operations.

→ Ch 3 GPU Architecture → Ch 14 Networking

---

### NVLS NVLink SHARP

In-network aggregation using compute units inside NVSwitch chips. Performs **AllReduce** summation in hardware as data flows through the switch fabric, freeing GPU SMs from aggregation work. Approximately 2× faster than Ring AllReduce for large messages (>1 MB). Auto-enabled on H100+ via NCCL ≥ 2.16 when NVSwitch hardware is present. NVLS reduces the AllReduce bandwidth burden from 2×(N-1)/N × msg to a single pass through the switch — at DP=8, this reduces AllReduce time by ~87% for large gradient tensors.

→ Ch 14 Networking

## O

2 TERMS

### Occupancy

The ratio of active warps to maximum possible warps per SM. High occupancy allows the warp scheduler to hide memory latency by switching to ready warps during HBM stalls. Low occupancy (caused by too many registers per thread, or large shared memory allocation per block) can reduce effective throughput. Measured with `ncu --metrics sm__warps_active.avg.pct_of_peak_sustained_active`. Note: high occupancy is not sufficient for performance — a compute-bound kernel with 50% occupancy may outperform a memory-bound kernel with 100% occupancy if the bottleneck is Tensor Cores, not warp scheduling.

→ Ch 7 Kernels

---

### OST Object Storage Target

An individual storage server in a **Lustre** filesystem, each contributing a fraction of total aggregate bandwidth. Typical AI cluster OST count: 40–200 per filesystem. Each OST serves a subset of stripes; distributing a file across more OSTs (via `lfs setstripe -c N`) increases I/O parallelism proportionally. A 128-OST Lustre filesystem with 4 GB/s per OST provides 512 GB/s aggregate — sufficient to checkpoint a 70B model in ~0.5 seconds vs 140 seconds if writing to a single OST.

→ Ch 15 Storage I/O

# P

7 TERMS

### PagedAttention

A **KV cache** memory management system (vLLM, 2023) that stores KV in fixed-size non-contiguous physical blocks with a per-sequence page table — analogous to virtual memory paging in OS design. Eliminates internal fragmentation (reduces memory waste from ~80% to ~1% by not pre-allocating maximum-length contiguous KV buffers). Enables **Copy-on-Write** prefix sharing. Industry standard in vLLM, TRT-LLM, SGLang. The key insight: KV blocks need not be contiguous in physical memory — a page table indirection enables arbitrary allocation while maintaining logical sequence continuity.

→ Ch 11 KV Cache → Ch 6 Inference

---

### PFC Priority Flow Control

IEEE 802.1Qbb Ethernet feature that enables lossless delivery for specific traffic classes by sending PAUSE frames upstream when switch buffers fill. Required for **RoCEv2** to provide RDMA semantics (which require zero packet loss for correct operation). PFC must be carefully tuned: too aggressive causes PAUSE storms that halt all traffic; too lenient causes packet loss that forces RDMA retransmits. Use with Active Congestion Control (**DCQCN**) to prevent incast events at AI training scales. InfiniBand is inherently lossless and does not require PFC.

→ Ch 14 Networking

---

### Pipeline Parallelism PP

A distributed training strategy that assigns different layers to different GPUs and pipelines microbatches through the stages. Enables training models with far more layers than fit on a single GPU's HBM. Communication: point-to-point activation tensors between adjacent pipeline stages (much lower bandwidth than all-reduce — enables cross-node PP over InfiniBand). Efficiency cost: see **Bubble Fraction**. 1F1B (interleaved 1-forward-1-backward) scheduling reduces bubble vs naive GPipe. Must pair with large microbatch counts ($M \geq 4 \times PP$) to achieve acceptable efficiency.

→ Ch 10 Training → Ch 14 Networking

---

### PPO Proximal Policy Optimization

The RL algorithm most commonly used for **RLHF**. Requires 4 concurrent model instances: rollout workers (inference via actor model), reward scorer, reference model (frozen SFT baseline), and policy updater. Infrastructure complexity: 3–4× model memory, coordinated inference+training serving, reward model inference pipeline. More powerful than **DPO** for complex reasoning tasks (e.g., math, code) but significantly higher infrastructure cost. OpenAI GPT-4 and Anthropic Claude use PPO-based RLHF; many open-source models use DPO as a simpler alternative.

→ Ch 10 Training

---

### Prefill

The phase of **inference** that processes the input prompt. Runs as a large batch GEMM on all prompt tokens simultaneously (the full S×d_model computation in one forward pass). Compute-bound at large batch/sequence sizes — the GEMM is large enough to saturate Tensor Cores. Duration determines **TTFT**: TTFT ≈ prefill_time + scheduling_queue_time. A 1,000-token prompt on H100 with Llama-3 70B takes ~150ms — much of TTFT in production comes from batching and queuing, not the compute itself.

→ Ch 6 Inference → Ch 11 KV Cache

---

### Prefix Caching

An inference optimization that stores KV cache blocks for common prefixes (system prompts, RAG documents) and reuses them across requests without recomputation. Reduces TTFT by 60–89% for workloads with fixed prefixes — the KV computation for the system prompt is done once and shared across all requests. Reduces KV memory by up to 91% for chatbot applications where a 2,000-token system prompt dominates sequence length. Implemented in vLLM via **RadixAttention** (tree-structured prefix sharing), TRT-LLM, and SGLang.

→ Ch 11 KV Cache → Ch 6 Inference

---

### PTQ Post-Training Quantization

Quantization applied after training is complete, using only a calibration dataset (128–512 samples). No retraining or fine-tuning required. **GPTQ** and **AWQ** are PTQ methods. Can be applied in minutes to hours vs days for **QAT**. Quality: generally <2% accuracy delta for INT4 on benchmarks like MMLU, Hellaswag. PTQ is the default production choice for quantization — the minimal quality delta vs significant inference cost reduction (2× throughput, 2× capacity) makes it economically dominant for most serving use cases.

→ Ch 8 Quantization

## Q

3 TERMS

---

### QAT Quantization-Aware Training

Quantization that simulates quantization error during training via "fake quantization" operators (quantize to INT4/FP8, dequantize back to BF16, continue computation in BF16). Allows the model to adapt its weights to minimize quality loss under quantization. Produces consistently higher quality than **PTQ** — typically <0.5% MMLU delta for INT4 vs <1.5% for PTQ — at the cost of significant training compute (~10–20% of pretraining cost). Used by NVIDIA for TensorRT-LLM FP8 quantization of GPT-4 class models, where every tenth of a percent of quality matters at scale.

→ Ch 8 Quantization

---

### QLoRA

A parameter-efficient fine-tuning method combining **NF4** quantized base model with **LoRA** adapters. Enables fine-tuning 70B+ models on a single 48 GB GPU. Quality: ~93–94% of full fine-tune on instruction following benchmarks. Infrastructure: bitsandbytes library + PEFT. The key innovation: gradients pass through the NF4 base model weights (which remain frozen) to the BF16 LoRA adapters — the base model does not need to be dequantized, only its forward activations need to be in BF16 precision.

→ Ch 8 Quantization → Ch 10 Training

---

### QoS Quality of Service

**SLURM** scheduling tier configuration that sets resource limits, priority levels, and preemption rules per user class. Three standard tiers: high-priority (production, preempts research), normal (standard research), low-priority (exploratory, can be preempted). Used to guarantee production training jobs always have access to GPU allocation while allowing research jobs to fill remaining capacity via backfill scheduling. Configured via `sacctmgr add qos` with `Priority`, `MaxTRES`, and `Preempt` parameters.

→ Ch 16 Cluster Mgmt

## R

9 TERMS

### RadixAttention

**SGLang**'s tree-structured prefix caching system. Stores KV cache blocks in a radix tree (trie), enabling physical block sharing for any shared prefix, not just fixed system prompts. Shared blocks have a single physical copy with reference counting — a block is freed only when its reference count reaches zero. Eviction policy: leaves first (most recently used shared prefixes retained longest, like an LRU cache). Achieves 91% KV cache hit rate on chatbot workloads with diverse conversation histories. Generalizes prefix caching to arbitrary prefix trees without manual prefix configuration.

→ Ch 11 KV Cache → Ch 6 Inference

---

### RDMA Remote Direct Memory Access

A network transfer mechanism that allows a NIC to read from or write to remote GPU or CPU memory without involving the remote CPU — the NIC handles the transfer autonomously. Provides low latency (~1–2 µs one-way for IB HDR) and low CPU overhead (near zero CPU cycles for in-flight transfers). Used in InfiniBand and **RoCEv2**. Critical for collective operations in distributed training: RDMA enables the NIC to directly DMA from GPU **HBM** to remote GPU HBM, bypassing CPU memory entirely via **GPUDirect RDMA**.

→ Ch 14 Networking

---

### ReduceScatter

A collective that reduces (sums) tensors and scatters the result: each process ends up with a disjoint shard of the reduced total. Used in **ZeRO**/**FSDP** for gradient synchronization. Lower bandwidth cost than **AllReduce** for large models (each rank only receives its own gradient shard, vs AllReduce where each rank receives the full reduced gradient). The AllReduce can be decomposed as ReduceScatter + AllGather — this decomposition is what enables Ring-AllReduce and is the foundation of ZeRO Stage 2/3 memory reduction.

→ Ch 10 Training → Ch 14 Networking

---

### Ridge Point

The **Arithmetic Intensity** at which the **Roofline** peak compute line meets the peak bandwidth line: `ridge = peak_FLOPS / peak_BW`. H100 SXM: 989 TFLOPS / 3.35 TB/s ≈ 295 FLOP/byte. Operations with AI < 295 on H100 are bandwidth-bound; AI > 295 are compute-bound. The ridge point shifts with quantization: H100 FP8 ridge = 1,978 TFLOPS / 3.35 TB/s ≈ 590 FLOP/byte — a GEMM compute-bound at BF16 may become bandwidth-bound if migrated to FP8 without increasing batch size proportionally.

→ Ch 1 Mindset → Ch 7 Kernels

---

### RLHF Reinforcement Learning from Human Feedback

The process of fine-tuning a pretrained LLM using human preference data to align outputs with human values. Standard pipeline: (1) SFT — supervised fine-tune on curated demonstrations; (2) Reward Model — train a classifier on chosen/rejected pairs; (3) RL — optimize policy against reward model via **PPO** or **DPO**. The reward model must be the same size as the policy (or larger) for reliable signal. Infrastructure cost of full RLHF with PPO: 3–4× the serving cost of the base model due to simultaneous reference model inference.

→ Ch 10 Training

---

### RMSNorm

A normalization layer that divides activations by their root-mean-square: `RMSNorm(x) = x / RMS(x) * γ` where `RMS(x) = sqrt(mean(x²))`. Faster than LayerNorm (one HBM pass vs two) because it skips mean subtraction. Standard in LLaMA-3, Mistral, Qwen. The primary fusion opportunity in transformer FFN layers: RMSNorm + GEMM can be fused into a single kernel, eliminating one full HBM read/write of the activation tensor. At 4K sequence length, 80 layers: fusion saves ~1.6 GB of HBM traffic per forward pass.

→ Ch 7 Kernels → Ch 9 Compilers

---

### RoCEv2 RDMA over Converged Ethernet v2

**RDMA** over standard Ethernet fabric. Lower cost than InfiniBand at equivalent bandwidth (400 GbE NICs at ~1/3 the cost of HDR IB). Requires **PFC** for lossless delivery and active congestion control (**DCQCN**) to prevent incast collapse in fat-tree topologies. Used by Meta for Llama-3 training at 24,576 H100 scale, and by most hyperscalers building proprietary AI training fabrics. The operational complexity of PFC/DCQCN tuning is the primary downside vs IB's simpler lossless architecture.

→ Ch 14 Networking

---

### Roofline Model

A performance model that characterizes kernel performance by plotting achievable FLOPS/s vs arithmetic intensity on log-log axes. The "roof" has two segments: bandwidth-bound slope (slope = HBM_BW, units FLOPS/s per FLOP/byte = bytes/s) and compute-bound flat ceiling (= peak_FLOPS). The key principle: performance is limited by the minimum of what bandwidth can feed and what compute can process. Used to classify every operation as bandwidth- or compute-bound, identify the correct optimization lever, and set theoretical performance targets. The central analytical framework of this book.

→ Ch 1 Mindset → Ch 7 Kernels → Ch 12 Benchmarking

---

### RoPE Rotary Positional Embedding

A position encoding method that rotates query and key vectors by a complex rotation corresponding to their position: `q_m = q * e^(imθ)`, `k_n = k * e^(inθ)`. The dot product q_m · k_n naturally encodes relative position (m-n) because the rotation cancels in the dot product. Enables relative position encoding that generalizes better to longer sequences than learned absolute positions. Standard in LLaMA-3, Mistral, Qwen. YaRN and LongRoPE extend context length by modifying the θ base — enabling 128K–1M+ context without full retraining.

→ Ch 2 Transformer

## S

5 TERMS

### Scaling Efficiency

The ratio of multi-GPU throughput to ideal linear scaling: `η(N) = throughput(N GPUs) / (N × throughput(1 GPU))`. Target: 85%+ at 64 GPUs, 74%+ at 1,024 GPUs on H100 clusters. Below 80% at 64 GPUs: almost always a configuration problem (NIC affinity, NCCL topology, gradient accumulation frequency). At 1,024 GPUs, scaling efficiency of 74% means 260 GPUs worth of compute is lost to communication and synchronization overhead — worth ~$780K/month at $3/GPU-hr. Every 1% efficiency improvement = ~$1M/year at that scale.

→ Ch 10 Training → Ch 12 Benchmarking → Ch 14 Networking

---

### SHARP **Scalable Hierarchical Aggregation and Reduction Protocol**

InfiniBand switch in-network AllReduce. Aggregation compute units inside Quantum2 switches perform gradient summation as packets flow through the fabric, without returning to GPU SMs. Reduces AllReduce traffic by ~50%, latency by ~30–40% for large messages. Available on IB only (not RoCEv2). At 1,024 GPU scale with SHARP, the AllReduce wall-time for a 1 GB gradient tensor drops from ~100ms (Ring) to ~60ms — recovering ~3% of compute time that would otherwise be idle during synchronization.

→ Ch 14 Networking

---

### SLURM

Simple Linux Utility for Resource Management. The dominant job scheduler for HPC and large-scale AI training. Three core daemons: `slurmctld` (controller — job queue and scheduling decisions), `slurmdbd` (accounting database — usage tracking for fair-share), `slurmd` (per-node agent — job launch and monitoring). Plugins for GPUs, InfiniBand topology, Lustre burst buffer, and cloud burst. At 1,000+ GPU scale, SLURM's controller can become a performance bottleneck — tune with `SchedulerParameters=sched_min_interval=2000` to reduce scheduling overhead.

→ Ch 16 Cluster Mgmt

---

### Speculative Decoding

An inference optimization that uses a cheap **draft model** to propose K candidate tokens, then verifies them all with one target model forward pass in parallel. Acceptance criterion: `min(1, p_target/p_draft)`. Mathematically exact: output distribution is identical to the target model (no quality degradation). Speedup: 2–3.5× for code and SQL workloads where the draft model (e.g., a 7B version of the 70B target) has high acceptance rate. Near-zero speedup for creative writing where token predictability is low. Requires maintaining two models in HBM simultaneously.

→ Ch 13 Speculative Decoding → Ch 6 Inference

---

### SwiGLU

A gated linear unit activation function: `SwiGLU(x) = SiLU(W₁x) ⊙ (W₃x)`. Used in LLaMA FFN layers. Requires two GEMMs ($W_1$ for the gate, $W_3$ for the linear path) followed by elementwise multiply, then one more GEMM ($W_2$) for the down-projection. $W_1$ and $W_3$ can be fused into a single wider GEMM (concatenate along output dimension). Standard in LLaMA-3, Mistral, Qwen. FFN dimension is typically 8/3 × d_model rather than 4× to maintain parameter count equivalent to a dense FFN layer.

→ Ch 2 Transformer → Ch 7 Kernels

## T

8 TERMS

### Tensor Core

Dedicated hardware units on NVIDIA GPUs that perform matrix multiply-accumulate in a single clock cycle: D = A × B + C where A, B, C, D are small matrix tiles. H100 achieves 989 TFLOPS BF16 using Tensor Cores vs ~20 TFLOPS for FP32 scalar units — a 50× compute density difference. Tensor Core utilization requires: (1) matrix dimensions aligned to multiples of 16 (BF16) or 8 (FP8); (2) correct memory layout (row-major for A, column-major for B); (3) warp groups configured for WGMMA (Hopper). Unaligned dimensions cause the compiler to pad with zeros, reducing effective FLOP utilization.

→ Ch 3 GPU Architecture → Ch 7 Kernels

### Tensor Parallelism TP

A parallelism strategy that splits individual weight matrices across GPUs: column-parallel for FFN gate/up projections (split output dimension), row-parallel for FFN down and attention output projections (split input dimension). Requires one AllReduce per transformer layer. Must be confined within the **NVLink** domain (TP ≤ 8) to avoid InfiniBand latency bottleneck: per-layer AllReduce at TP=8 takes ~1µs via NVLink vs ~5–10µs via IB — the difference compounds to >10% throughput loss at TP=8 cross-node. Megatron-style TP is the standard reference implementation.

→ Ch 10 Training → Ch 14 Networking

### TF32 TensorFloat 32

A 19-bit compute format (8 exponent + 10 mantissa, stored in 32-bit registers) used transparently by A100+ Tensor Cores when FP32 input is provided. Delivers ~16× FP32 throughput with ~1% accuracy loss on typical ML workloads — the truncated mantissa rarely matters because weight gradients are far noisier than the precision reduction. Enabled by default in PyTorch 1.7+ via `torch.backends.cudnn.allow_tf32 = True`. Not a user-visible format — it's an internal Tensor Core optimization that happens automatically without code changes.

→ Ch 8 Quantization → Ch 3 GPU Architecture

---

### TorchInductor

The backend compiler in PyTorch 2.x (`torch.compile`) that converts traced FX computation graphs into optimized Triton kernels. Performs operator fusion, layout optimization (NCHW vs NHWC), and tile size autotuning. Targets NVIDIA (via Triton → PTX), AMD (via Triton → HSACO), and CPU. Typical speedup vs eager mode: 20–40% for transformer inference, 10–20% for training. The compilation overhead (first-run JIT) is amortized over subsequent calls. `torch.compile(model, mode="max-autotune")` exhaustively searches tile sizes — adds 10 min compilation for 10–15% additional improvement.

→ Ch 9 Compilers

---

### TPOT Time Per Output Token

The time between consecutive tokens during the **decode** phase. Primary latency metric for users during streaming generation — directly determines the perceived "typing speed" of the model. Formula at small batch: `TPOT ≈ model_bytes / HBM_bandwidth`. H100 with 70B BF16: 140 GB / 3,350 GB/s ≈ 42ms/token = 24 tokens/sec. Target SLA: <50ms/token for natural reading pace, <33ms/token for fast reading. TPOT scales linearly with model size and inversely with HBM bandwidth — the direct case for quantization ROI.

→ Ch 6 Inference → Ch 12 Benchmarking

---

### Transformer Engine TE

NVIDIA's library implementing automatic **FP8** precision management for transformer layers. Maintains per-tensor rolling activation maximums (`amax_history` over last 16 steps), computes dynamic per-tensor scaling factors, performs FP8 GEMM on Tensor Cores, and descales outputs back to BF16/FP32 for accumulation. Enables FP8 training and inference with a single API wrapper change: `te.Linear` instead of `nn.Linear`. Used by NVIDIA Megatron-LM, Llama implementations, and most major training frameworks as of 2024.

→ Ch 8 Quantization → Ch 10 Training

---

### Triton

An open-source Python-level GPU programming language (OpenAI, 2021). Abstracts shared memory management, warp scheduling, and autotuning behind a Python API. A Triton kernel expresses computation in terms of tiles and memory access patterns; the compiler handles bank conflict avoidance, register allocation, and instruction scheduling. Used by TorchInductor for code generation and by engineers for custom GPU kernels without full CUDA C++ expertise. Triton kernels are typically 80–95% of the performance of hand-tuned CUDA for memory-bandwidth-bound operations; competitive for compute-bound operations at standard tile sizes.

→ Ch 7 Kernels → Ch 9 Compilers

---

### TTFT Time to First Token

The latency from request submission to generation of the first output token. Dominated by **prefill** computation time plus queuing latency. The primary user-perceived latency for interactive LLM services — users notice TTFT >500ms as "slow." Production target: P50 <300ms, P99 <800ms for chatbot applications. P99 TTFT is the most important SLO metric to monitor — P50 can be acceptable while P99 spikes indicate queue backup or **KV cache cascade**. Diagnose P99 TTFT spikes via request queue depth and KV cache utilization correlation.

→ Ch 6 Inference → Ch 12 Benchmarking → Ch 17 Telemetry

## V

2 TERMS

---

### vLLM

An open-source high-performance LLM serving library (UC Berkeley, 2023). Pioneered **PagedAttention**, which eliminated KV cache memory fragmentation and became the industry standard for production inference. Current features (2024–2025): continuous batching, prefix caching, speculative decoding, FP8 KV cache quantization, disaggregated prefill-decode. The de facto reference implementation for production LLM serving — used by most teams as the baseline before profiling and customizing. Integrates with Ray for multi-node distributed serving.

→ Ch 6 Inference → Ch 11 KV Cache

---

### Volcano

A Kubernetes batch scheduling framework that adds gang scheduling (all pods in a job start together or none start), queue management, and priority preemption for distributed ML training jobs. Prevents GPU deadlocks from partial rank allocation — without gang scheduling, a 256-GPU training job might start with 200 GPUs allocated, hold those GPUs while waiting for the remaining 56, and deadlock with other jobs doing the same. Required for any Kubernetes-based ML platform running large training jobs. Used by Alibaba, ByteDance, and Huawei for large-scale training cluster orchestration.

→ Ch 16 Cluster Mgmt

## W

2 TERMS

### Warp

The fundamental CUDA execution unit: 32 threads that execute the same instruction simultaneously on an SM via the SIMT (Single Instruction, Multiple Thread) model. All 32 threads in a warp must execute the same instruction at each clock cycle. Control flow divergence (if-else with different paths per thread) causes serialization: both branches execute, with inactive threads masked — cutting effective throughput by 2× or more. Warps are the scheduling unit for the SM warp scheduler: 4 warp schedulers per SM on H100, each dispatching up to 2 independent warps per cycle for latency hiding.

→ Ch 3 GPU Architecture → Ch 7 Kernels

### Wavefront

AMD's equivalent of a CUDA warp, but with 64 threads (2× NVIDIA). Executes as a single logical instruction stream across 64 SIMD lanes. The 2× width affects kernel design: tile sizes that are optimal on NVIDIA (multiples of 32) may not be optimal on AMD (multiples of 64). Reduction loop depths must account for wavefront width: a warp-level reduction over 32 threads takes $\log_2(32)=5$ stages; over 64 threads takes 6 stages. MI300X uses 4 SIMD-16 units per CU, each processing 16 lanes of a 64-thread wavefront per cycle.

→ Ch 3 GPU Architecture

## X

2 TERMS

### XID Error

NVIDIA GPU error code logged to the kernel message buffer (`dmesg | grep XID`). Critical production codes: XID 48 (uncorrectable double-bit ECC error — replace GPU immediately); XID 95 (uncontained error / engine reset failure — take node offline); XID 74 (NVLink error — investigate cable/connector). Non-critical: XID 63 (row remap — normal aging, monitor for acceleration). Monitoring pipeline: DCGM → Prometheus → alert at first XID 48/95. Build automated node eviction triggered by XID 95 to prevent corrupted training runs from continuing on a dying GPU.

→ Ch 17 Telemetry

---

### XLA **Accelerated Linear Algebra**

Google's ahead-of-time compiler for JAX/TensorFlow models. Compiles from HLO (High-Level Operations) IR to PTX (NVIDIA), HLO-lowered assembly (AMD), or TPU assembly. All tensor shapes must be statically known at compile time — dynamic shapes require recompilation or padding to fixed buckets. The only compiler supported on Google TPUs. XLA's ahead-of-time compilation produces higher-quality kernel code than JIT compilers for stable workloads, at the cost of long compilation times (minutes to hours for large models) and rigid shape requirements.

→ Ch 9 Compilers

## Z

1 TERM

### ZeRO **Zero Redundancy Optimizer**

DeepSpeed's memory sharding framework with three stages. Stage 1: shard optimizer state (8× AdamW: 4× param + 2× momentum + 2× variance) across DP ranks. Stage 2: + shard gradients. Stage 3: + shard model parameters. Stage 3 (= **FSDP**) enables models of arbitrary size across any number of GPUs. Memory per GPU: model_params × 16 bytes / (DP_degree) for parameters alone. Communication cost: ZeRO-3 requires 3× the AllReduce bandwidth of naive DP (AllGather for params before forward, ReduceScatter for gradients after backward, AllGather for params again for optimizer step) — plan network bandwidth accordingly.

→ Ch 10 Training → Ch 14 Networking

## Quick Acronym Index

*All abbreviations used in this book, with full expansion. Terms with dedicated glossary entries are in bold.*

| Acronym | Full Name | Primary Chapter |
|---|---|---|
| **AI** | Arithmetic Intensity (FLOPs/byte) | Ch 1, 7, 12 |
| **AWQ** | Activation-Aware Weight Quantization | Ch 8 |
| **BF16** | BrainFloat16 | Ch 3, 8 |
| **CP** | Context Parallelism | Ch 10, 14 |
| **CTE** | Correctable Threshold Exceeded (ECC) | Ch 17 |
| **DCGM** | Data Center GPU Manager | Ch 17 |
| **DCQCN** | Data Center Quantized Congestion Notification | Ch 14 |
| **DCE** | Dead Code Elimination (compiler) | Ch 9 |
| **DCP** | Distributed CheckPoint (PyTorch) | Ch 10 |
| **DP** | Data Parallelism | Ch 10 |
| **DPO** | Direct Preference Optimization | Ch 10 |
| **ECC** | Error Correction Code | Ch 3, 17 |
| **EP** | Expert Parallelism | Ch 13, 14 |
| **FP8** | 8-bit Floating Point (E4M3 / E5M2) | Ch 8, 10 |
| **FSDP** | Fully Sharded Data Parallel | Ch 10 |
| **GEMM** | General Matrix Multiplication | Ch 7 |
| **GQA** | Grouped Query Attention | Ch 2, 11 |
| **GPTQ** | Second-order PTQ quantization method | Ch 8 |
| **HBM** | High-Bandwidth Memory | Ch 3, 4 |
| **HFU** | Hardware FLOP Utilization | Ch 10, 12 |
| **IB** | InfiniBand | Ch 14 |
| **KV** | Key-Value (cache) | Ch 11 |
| **LoRA** | Low-Rank Adaptation | Ch 10 |
| **MFU** | Model FLOP Utilization | Ch 10, 12 |
| **MHA** | Multi-Head Attention | Ch 2, 11 |
| **MIG** | Multi-Instance GPU | Ch 3, 6 |
| **MLA** | Multi-Head Latent Attention | Ch 2, 11, 13 |

| | | |
|---|---|---|
| **MoE** | Mixture of Experts | Ch 13 |
| **MQA** | Multi-Query Attention | Ch 2, 11 |
| **NCCL** | NVIDIA Collective Communications Library | Ch 14 |
| **NF4** | NormalFloat 4-bit | Ch 8 |
| **NVFP4** | NVIDIA FP4 (Blackwell) | Ch 3, 8 |
| **NVLS** | NVLink SHARP (in-network aggregation) | Ch 14 |
| **OST** | Object Storage Target (Lustre) | Ch 15 |
| **PFC** | Priority Flow Control | Ch 14 |
| **PP** | Pipeline Parallelism | Ch 10 |
| **PPO** | Proximal Policy Optimization | Ch 10 |
| **PTQ** | Post-Training Quantization | Ch 8 |
| **QAT** | Quantization-Aware Training | Ch 8 |
| **RDMA** | Remote Direct Memory Access | Ch 14 |
| **RLHF** | Reinforcement Learning from Human Feedback | Ch 10 |
| **RMSNorm** | Root Mean Square Normalization | Ch 2, 7 |
| **RoPE** | Rotary Positional Embedding | Ch 2 |
| **SDC** | Silent Data Corruption | Ch 17 |
| **SLA** | Service Level Agreement | Ch 6, 12 |
| **SM** | Streaming Multiprocessor | Ch 3 |
| **SMEM** | Shared Memory (on-chip SRAM per SM) | Ch 3, 7 |
| **SWA** | Sliding Window Attention | Ch 2 |
| **TE** | Transformer Engine (NVIDIA FP8 library) | Ch 8, 10 |
| **TF32** | TensorFloat 32 | Ch 8 |
| **TP** | Tensor Parallelism | Ch 10, 14 |
| **TPOT** | Time Per Output Token | Ch 6, 12 |
| **TRT** | TensorRT (NVIDIA inference optimizer) | Ch 9 |
| **TTFT** | Time to First Token | Ch 6, 12, 17 |
| **XID** | NVIDIA GPU Error Code | Ch 17 |

| | | |
|---|---|---|
| **XLA** | Accelerated Linear Algebra (Google) | Ch 9 |
| **ZeRO** | Zero Redundancy Optimizer (DeepSpeed) | Ch 10 |