APPENDIX C

Reading Roadmap: Papers, Tools & Continued Learning

The canonical papers every AI infrastructure engineer must read · The tools used in production · A 12-month learning plan from first principles to Principal level

---

HOW TO USE THIS APPENDIX

Papers are rated by priority:

MUST READ

(career-defining, foundational),

IMPORTANT

(deep expertise), and

CONTEXT

(breadth). Read all MUST READ papers before any principal-level interview. Tools are organized by use case. The 12-month plan assumes 5–8 hours/week of dedicated study.

1 · Foundational Papers — GPU Architecture & Performance

★ MUST READ

Attention Is All You Need

Vaswani et al. · Google Brain · NeurIPS 2017

The original transformer paper. Every LLM architecture descends from this. Understand the QKV attention mechanism, multi-head formulation, and positional encoding from the source. Read the math, not summaries.

```
Key formula: Attention(Q,K,V) = softmax(QKᵀ/√d_k)V
```

★ MUST READ

Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures

Williams, Waterman, Patterson · UC Berkeley · CACM 2009

The foundational performance model for this book. The roofline diagram, arithmetic intensity, and ridge point concept come from here. Short paper — read the original. Every performance claim in AI infrastructure rests on this framework.

```
Key concept: Attainable performance = min(F_peak, AI × BW)
```

★ MUST READ

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Dao et al. · Stanford · NeurIPS 2022

The single most impactful algorithmic paper for LLM training and inference. Understand the tiling approach, why it reduces HBM traffic from $O(N^2)$ to $O(N)$, and how it enables long-context training. FlashAttention-2 and -3 are incremental improvements — start here.

```
Key insight: Avoid materializing the N×N score matrix in HBM
```

★ MUST READ

Efficient Large Scale Language Modeling with Mixtures of Experts

Artetxe et al. · Meta AI · 2021 · Also: Switch Transformer (Fedus et al., 2021)

MoE is the dominant architecture for frontier models. Understand the routing mechanism, expert capacity, load balancing, and why MoE enables parameter scaling without proportional compute increase. Essential for understanding GPT-4, Mixtral, and Grok.

```
Key concept: Sparse conditional computation — route each token to top-K of N experts
```

IMPORTANT

NVIDIA H100 Tensor Core GPU Architecture (Hopper Architecture Whitepaper)

NVIDIA · 2022 · Available at developer.nvidia.com

The authoritative hardware reference. Understand the SM architecture, NVLink topology, Transformer Engine, FP8 support, and GH100 die organization. This is what you cite when someone asks how H100 achieves 989 TFLOPS.

```
Key specs: 132 SMs, 989 TFLOPS BF16, 3.35 TB/s HBM3, 900 GB/s NVLink
```

IMPORTANT

AMD CDNA3 / MI300X Architecture Whitepaper

AMD · 2023 · Available at amd.com/developer

MI300X is the primary H100 competitor with a distinct chiplet architecture (6 XCDs + 4 MCDs). Understanding the differences in memory topology, unified HBM access, and the absence of NVLink informs when MI300X outperforms H100 (large memory workloads, inference).

```
Key specs: 192 GB HBM3, 5.3 TB/s BW, 1307 TFLOPS BF16, 6 XCDs
```

2 · Inference Systems Papers

★ MUST READ

Efficient Memory Management for Large Language Model Serving with PagedAttention

Kwon et al. · UC Berkeley · SOSP 2023

Introduces PagedAttention — the foundation of vLLM. Understand the block table, physical/logical page mapping, and why it achieves 2–4× higher throughput than Hugging Face. Essential for anyone designing or evaluating LLM serving systems.

```
Key insight: Apply OS virtual memory paging to KV cache management
```

★ MUST READ

Orca: A Distributed Serving System for Transformer-Based Generative Models

Yu et al. · OSDI 2022

Introduces continuous batching (iteration-level scheduling). Understand why static batching wastes GPU resources and how continuous batching increases throughput 30–36× vs naive batching. This paper unlocked production-scale LLM serving.

```
Key concept: Preempt and re-queue requests at the iteration level, not request level
```

★ MUST READ

Fast Inference from Transformers via Speculative Decoding

Leviathan et al. · Google · ICML 2023

The foundational speculative decoding paper. Understand the draft-then-verify loop, the acceptance criterion (preserving target distribution exactly), and the speedup formula. Every major inference system now implements some variant.

```
Key formula: Speedup = (1 + α + α² + ... + αᵞ) / (1 + γ/r)
```

IMPORTANT

DistServe: Disaggregating Prefill and Decoding for Goodput-Optimized Large Language Model Serving

Zhong et al. · OSDI 2024

Introduces prefill/decode disaggregation: run prefill on compute-optimized nodes, decode on memory-bandwidth-optimized nodes. Challenges the assumption that one GPU serves both phases. Foundational for next-generation serving architectures.

IMPORTANT

GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

Ainslie et al. · Google · EMNLP 2023

Introduces Grouped Query Attention — used in LLaMA-2/3, Mistral, and virtually every modern open model. Understand why GQA reduces KV cache size by 8–16× with minimal quality loss. Critical for memory sizing formulas.

```
Key insight: Share K,V tensors across groups of Q heads → 8× KV cache reduction
```

3 · Distributed Training Papers

★ MUST READ

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism

Shoeybi et al. · NVIDIA · 2019

Introduces tensor parallelism as implemented in every major training system. Understand how attention heads and FFN layers are split across GPUs, where AllReduce happens, and why this requires NVLink-class bandwidth. Megatron-LM is the reference implementation.

```
Key concept: Column-parallel + row-parallel linear layers with one AllReduce per layer pair
```

★ MUST READ

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

Rajbhandari et al. · Microsoft DeepSpeed · SC 2020

ZeRO-1/2/3 are the foundation of large-scale training. Understand the three stages of state sharding, the memory savings formula, and why ZeRO-3 (≈ PyTorch FSDP) enables training models that don't fit on any single GPU or node.

```
Key formula: Memory per GPU = Total_states / N_GPUs (ZeRO-3)
```

★ MUST READ

Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM

Narayanan et al. · NVIDIA · SC 2021

Introduces interleaved 1F1B pipeline scheduling and the 3D parallelism (TP + PP + DP) combination that enabled GPT-3 scale training. The pipeline bubble formula and the MFU analysis come from this paper. The benchmark numbers are still referenced today.

```
Key formula: Pipeline bubble fraction = (PP-1) / (PP-1 + m)
```

IMPORTANT

Scaling Laws for Neural Language Models

Kaplan et al. · OpenAI · 2020

Establishes the power-law relationship between model size, compute budget, and loss. Foundational for understanding why trillion-parameter models are not always better than compute-optimal smaller models. Chinchilla (Hoffmann et al., 2022) refines these laws — read both.

```
Key result: Compute-optimal training: N ∝ C^0.5 (model size scales as sqrt of compute)
```

IMPORTANT

GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers

Frantar et al. · IST Austria · ICLR 2023

The dominant post-training quantization method for LLMs. Understand the layer-wise quantization approach (OBQ), why it preserves accuracy at INT4/INT3, and when to use GPTQ vs AWQ vs direct FP8 quantization.

4 · Systems & Infrastructure Papers

★ MUST READ

MLPerf: A Broad ML Benchmark Suite for Measuring Performance of ML Software Frameworks, Processors, and Clouds

Mattson et al. · IEEE Micro 2020

MLPerf defines how AI hardware performance is measured and compared. Understand the Training and Inference suites, the metric definitions (time-to-train, queries per second), and why MLPerf results require careful interpretation. The standard everyone cites.

IMPORTANT

Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations

Tillet et al. · MIT · MAPL 2019

Triton is the primary tool for writing custom GPU kernels in Python. Understand the tile abstraction, auto-tuning, and why Triton kernels often match hand-tuned CUDA performance. FlashAttention-2 is implemented in Triton.

```
Toolchain: python + triton → compiles to PTX → runs on GPU without CUDA expertise
```

IMPORTANT

NCCL: A Communication Primitive Library for CUDA-Based Parallel Computing

NVIDIA · Technical Documentation + GitHub

Every distributed training job uses NCCL. Understand the AllReduce ring algorithm, AllGather, ReduceScatter, and the NVLink vs IB transport selection logic. Read the GitHub README and algorithm papers.

```
Key: github.com/NVIDIA/nccl + NCCL_DEBUG=INFO for algorithm visibility
```

CONTEXT

In-Datacenter Performance Analysis of a Tensor Processing Unit

Jouppi et al. · Google · ISCA 2017

The TPU-v1 architecture paper. Understand the systolic array design, why it differs from GPU SIMT, and why this shapes Google's model development choices. Essential context for Ch 3A (accelerator landscape).

CONTEXT

Pathways: Asynchronous Distributed Dataflow for ML

Barham et al. · Google Brain · MLSys 2022

Google's next-generation ML infrastructure for heterogeneous compute. Understand why they moved beyond the standard data-parallel training paradigm and how asynchronous execution changes the distributed systems design space.

5 · Production Profiling & Monitoring Tools

Nsight Systems

SYSTEM-LEVEL GPU TIMELINE PROFILING

CPU+GPU timeline traces. Shows kernel launch overhead, CUDA stream concurrency, NVLink utilization, and memory transfers. First tool to open when diagnosing performance.

```
nsys profile --stats=true python train.py
```

Nsight Compute

PER-KERNEL GPU MICRO-PROFILING

Roofline analysis, occupancy, memory bandwidth utilization, warp efficiency, L1/L2 hit rates. Use when Nsight Systems identifies a slow kernel.

```
ncu --set full -o profile.ncu-rep python kernel_test.py
```

DCGM (Data Center GPU Manager)

FLEET-LEVEL GPU HEALTH MONITORING

XID error detection, ECC tracking, power/temperature monitoring, NVLink error counters. Runs as daemon on every compute node. Integrates with Prometheus.

```
dcgmi dmon -e 203,204,1001,1002,1003
```

PyTorch Profiler

PYTHON-LEVEL TRAINING PROFILING

Operator-level timing, CUDA kernel attribution, memory allocation traces. Integrates with TensorBoard. Lower overhead than Nsight for training loop analysis.

```
with torch.profiler.profile(activities=[...]) as p:
```

Prometheus + Grafana

CLUSTER-LEVEL METRICS & DASHBOARDS

DCGM exporter pushes GPU metrics to Prometheus at 1s resolution. Grafana dashboards for MFU, memory utilization, power, temperature per node. The standard production monitoring stack.

```
dcgm-exporter --address 0.0.0.0:9400
```

ROCm Profiler (rocprof)

AMD GPU PROFILING

Equivalent to Nsight Compute for AMD MI300X/MI250. Hardware counters, kernel timing, memory access patterns. Essential for AMD GPU optimization work.

```
rocprof --stats python train.py
```

py-spy

PYTHON CPU PROFILER (LIVE)

Sample Python processes without modification. Critical for diagnosing NCCL hangs — py-spy dump shows which threads are stuck in ncclAllReduce.

```
py-spy dump --pid $(pgrep python)
```

nvidia-smi / rocm-smi

REAL-TIME GPU STATUS

First-look tool for GPU utilization, memory usage, temperature, power draw. The dmon subcommand enables continuous polling for monitoring scripts.

```
nvidia-smi dmon -s u,m,p,t -d 1
```

6 · Key Frameworks & Codebases to Study

Framework	What to Study	GitHub	Chapter Ref
vLLM	PagedAttention, continuous batching scheduler, worker/engine architecture	vllm-project/ vllm	Ch 6, 11
Megatron-LM	TP/PP/DP implementations, interleaved scheduling, communication primitives	NVIDIA/ Megatron-LM	Ch 10, 14
DeepSpeed	ZeRO-1/2/3, gradient compression, pipeline engine	microsoft/ DeepSpeed	Ch 10
PyTorch FSDP	ZeRO-3 equivalent, mixed precision, gradient checkpointing integration	pytorch/ pytorch	Ch 10, 19
FlashAttention	Tiled attention kernel in Triton/ CUDA, FA-2 backward pass	Dao-AILab/ flash-attention	Ch 4, 7
Triton	Tiled GPU programming, auto-tuning, matmul/softmax kernels	openai/triton	Ch 7, 9
NCCL	Ring-AllReduce algorithm, topology-aware routing, debug flags	NVIDIA/nccl	Ch 14
SGLang	Radix attention (prefix caching), structured generation, server architecture	sgl-project/ sglang	Ch 6, 11
TensorRT-LLM	Quantization, kernel fusion, NVIDIA-specific inference optimization	NVIDIA/ TensorRT-LLM	Ch 6, 8, 9
DCGM	GPU health monitoring, XID handling, Prometheus exporter	NVIDIA/DCGM	Ch 17, 19

7 · 12-Month Learning Plan: First Principles to Principal Level

> **PREREQUISITES**

Assumes: Python proficiency, basic neural network understanding, familiarity with Linux. Does NOT assume: GPU programming, distributed systems, or hardware knowledge. Plan requires 5–8 focused hours per week.

MONTHS 1–2 · FOUNDATION

GPU Architecture & The Roofline Model

- Read Ch 1 (Performance Mindset) and Ch 3 (GPU Architecture) of this book
- Read the Roofline paper (Williams et al., 2009) — derive the ridge point formula
- Read the NVIDIA H100 Architecture Whitepaper
- Lab: Run `nvidia-smi dmon` during a simple PyTorch workload. Watch SM utilization and memory bandwidth simultaneously
- Lab: Profile a matrix multiply with Nsight Compute. Find it on the roofline chart
- Lab: Calculate arithmetic intensity for GEMM at batch=1 vs batch=32. Verify with measurement
- Goal: Derive ridge point from spec sheet. Explain why GEMV is always memory-bound

MONTHS 3–4 · LLM INTERNALS

Transformer Architecture & Attention Mechanics

- Read "Attention Is All You Need" — implement self-attention from scratch in PyTorch
- Read Ch 2 (Transformer Architecture) of this book
- Read FlashAttention paper — understand the tiling argument
- Read the GQA paper — implement KV cache sizing formula for LLaMA-3
- Lab: Profile standard attention vs FlashAttention at S=1024, 4096, 8192. Measure memory usage
- Lab: Calculate KV cache size for LLaMA-3 70B at 4K, 32K, and 128K context
- Goal: Draw the transformer forward pass FLOP breakdown. Identify where memory vs compute dominates

MONTHS 5–6 · INFERENCE SYSTEMS

Production LLM Serving Architecture

- Read Ch 6 (Inference) and Ch 11 (KV Cache) of this book
- Read PagedAttention paper (Kwon et al., SOSP 2023)
- Read Orca continuous batching paper

- Lab: Deploy vLLM locally. Benchmark with varying batch sizes, measure TTFT and TPOT
- Lab: Implement PagedAttention block table logic from scratch (conceptual, ~100 lines Python)
- Lab: Measure throughput vs latency curve — find the knee of the curve
- Goal: Design a serving system for 10K RPM on LLaMA-3 70B — specify GPUs, TP, batching strategy

MONTHS 7–8 · DISTRIBUTED TRAINING

4D Parallelism & Large-Scale Training

- Read Ch 10 (Distributed Training) of this book
- Read Megatron-LM papers (2019, 2021) — focus on TP splitting and PP scheduling
- Read ZeRO paper — calculate memory savings for each stage
- Lab: Run a small training job with FSDP (ZeRO-3). Profile AllReduce time
- Lab: Calculate MFU for a known training run. Identify the largest efficiency gap
- Lab: Implement a simple ring-AllReduce in Python to internalize the algorithm
- Goal: Design parallelism strategy for 70B model pre-training on 1,024 H100s. Calculate MFU

MONTHS 9–10 · OPTIMIZATION TECHNIQUES

Quantization, Kernels, Fusion & Compilation

- Read Ch 7 (GPU Kernels), Ch 8 (Quantization), Ch 9 (Fusion/Compiler) of this book
- Read GPTQ paper. Run GPTQ quantization on a 7B model. Measure quality vs speed tradeoff
- Write a simple Triton kernel (vector add, then layer norm). Profile with Nsight Compute
- Lab: Use torch.compile on a training loop. Measure before/after. Inspect the compiled graph
- Lab: Fuse LayerNorm + GELU manually. Verify it reduces HBM traffic vs separate kernels
- Goal: Given a slow training step, identify if it is kernel-bound, memory-bound, or communication-bound. Fix it

MONTHS 11–12 · CLUSTER SCALE & OPERATIONS

Infrastructure, Telemetry & Principal Practice

- Read Ch 12 (Benchmarking), Ch 16 (Cluster Mgmt), Ch 17 (Telemetry), Ch 19 (Resilience)
- Set up DCGM + Prometheus + Grafana on a 1–4 GPU system. Build an MFU dashboard
- Read Ch 18 (Principal Career) — write your own technical story bank
- Practice: Do 5 system design mock interviews. Design: LLM serving at 1M RPM, distributed training at 10K GPU scale, checkpoint strategy for 72-hour run
- Practice: Read 2 recent MLSys / OSDI / SOSP papers. Summarize the performance insight in one paragraph
- Goal: Complete a full performance root cause analysis on a real system. Write up the methodology

8 · Staying Current — Sources & Strategies

Primary Sources (Weekly Reading)

Source	What to Follow	Signal-to-Noise
arXiv cs.LG + cs.DC	MLSys papers, infrastructure techniques, new attention variants	High — primary source
NVIDIA Developer Blog	Hopper/Blackwell architecture notes, TensorRT updates, NVLink deep dives	High — authoritative
AMD ROCm Blog	MI300X optimization guides, CDNA architecture updates	High for AMD
vLLM / SGLang GitHub Issues	Real production problems, performance regression discussions	Very high — practitioner-level
Meta AI Research blog	LLaMA releases, training infrastructure at scale	High
Hugging Face Blog	Practical optimization posts, new model architecture summaries	Medium — varies by author
	Practitioner insights, new paper reactions	Medium — filter signal

Source	What to Follow	Signal-to-Noise
Twitter/X: @karpathy, @timdettmers, @pommeranch		

Conferences to Track (Read Proceedings Annually)

MLSys

SYSTEMS FOR MACHINE LEARNING

The primary venue for AI infrastructure papers. vLLM, FlashAttention-2, Alpa, and most serving system papers appear here first. Read all accepted papers annually.

OSDI / SOSP / EuroSys

OPERATING SYSTEMS / DISTRIBUTED SYSTEMS

PagedAttention, Orca, DistServe appeared at OSDI/SOSP. Distributed inference and training systems increasingly appear here as the field matures.

ISCA / MICRO / HPCA

COMPUTER ARCHITECTURE

GPU architecture papers, accelerator designs, memory system innovations. The TPU paper appeared at ISCA. Follow for next-generation hardware understanding.

SC / ICS / PPoPP

HIGH-PERFORMANCE COMPUTING

MPI, collective communication algorithms, large-scale parallel programming. ZeRO appeared at SC. Understanding HPC roots is increasingly relevant.

The Meta-Skill: Reading Papers Efficiently

THREE-PASS METHOD FOR AI INFRASTRUCTURE PAPERS

How to absorb a paper in 30–60 minutes

- **Pass 1 (5 min):** Read title, abstract, introduction last paragraph, conclusion. Understand: what problem, what claim, what result.
- **Pass 2 (20 min):** Read figures, tables, and section headers. Look at benchmark results. Understand: how they measured, what baseline they beat, by how much.
- **Pass 3 (30 min):** Read the methods section. Identify the key formula or algorithm. Understand *why* it works, not just *that* it works.
- **Write one paragraph:** The problem, the insight, the result, and one open question. If you cannot write this, you have not understood the paper.

THE PRINCIPAL ARCHITECT'S READING DISCIPLINE

Read 2 papers per week from arXiv or conference proceedings. After each paper, write a one-paragraph summary: problem, insight, result, open question. Build a personal notes

> system indexed by topic (inference, training, memory, networking). In 12 months you will have read over 100 papers and have a searchable knowledge base that most engineers do not. That is the compounding advantage of consistent reading.
