The Accelerator Landscape: Beyond GPUs

TPUs, Custom Silicon, Neural Engines — Why the World Needs More Than NVIDIA

AI/ML Infrastructure from Silicon to Scale · Venkat Vinjam

> *"A GPU is a general-purpose parallel processor that happens to be excellent at matrix multiply. A TPU is a matrix multiply processor that happens to be able to run general programs. The difference in that sentence is worth billions of dollars."*
>
> — AI Infrastructure Design Principle

CHAPTER OBJECTIVES — BY THE END OF THIS CHAPTER YOU WILL BE ABLE TO:

- Explain the systolic array architecture of Google TPUs and contrast it with GPU SIMT
- Describe the Apple Neural Engine and its role in on-device AI
- Evaluate AWS Trainium and Inferentia against GPU alternatives with TCO math
- Understand Intel Gaudi 3's positioning and differentiation
- Explain Meta's MTIA architecture and why recommendation models need custom silicon
- Articulate Groq's LPU architecture and the deterministic latency promise
- Apply a structured decision framework for accelerator selection

NVIDIA GPUs command 80–90% of the AI accelerator market by revenue — but 100% of the world's AI inference does not run on NVIDIA hardware. Google trains Gemini on TPUs. Apple runs on-device LLMs on the Neural Engine inside every iPhone. AWS customers deploy billions of daily inferences on Inferentia chips. Meta serves its recommendation models on custom MTIA silicon. Groq has redefined what "deterministic latency" means for LLM serving.

A Principal Performance Architect who only knows GPUs is half-equipped. This chapter provides the architectural literacy needed to reason about the full accelerator landscape — when to choose each, how each differs from GPU execution, and what each teaches us about the fundamental trade-offs in AI hardware design.

3A.1 Why Custom Accelerators Exist

3A.1.1 The GPU Tax

GPUs are designed for a broad class of parallel workloads: scientific simulation, graphics rendering, data analytics, and AI. This generality is expensive. Every die area square millimeter devoted to rasterization hardware, texture units, and display engines is die area *not* devoted to AI compute.

```
WHAT A GPU CARRIES THAT DEDICATED AI SILICON DOESN'T:

Feature                GPU Uses It For        AI Inference Needs It?
──────────────────────────────────────────────────────────────────────
Rasterization HW       Graphics pipelines     NO
Texture units          Image rendering        NO
Display engines        Monitor output         NO
General SIMT cores     Flexible parallelism   SOMETIMES
Complex caches         Irregular workloads    PARTIALLY
PCIe PHY overhead      Host connectivity      SOMETIMES
Multi-tenant vGPU      Cloud sharing          SOMETIMES

Estimated die area wasted on non-AI features: 15–30%
Estimated power wasted at typical AI inference workloads: 10–20%

A dedicated AI ASIC uses 100% of die area for the one thing it does:
matrix multiply + memory access for a specific workload class.
```

3A.1.2 The Three Motivations for Custom Silicon

Cost at scale: At hyperscaler inference volumes, a 30% cost reduction from custom silicon is worth billions annually. Google's TPU fleet serves trillions of queries per day — even a 10% cost advantage over equivalent GPU capacity is immense.

Workload specialization: Recommendation models have fundamentally different compute characteristics than transformer LLMs — sparse embedding lookups, irregular memory access, relatively small matrix multiplies. GPUs are poorly matched to these workloads.

Supply chain independence: Dependence on a single supplier creates strategic vulnerability. Companies with the engineering resources to build custom silicon insulate themselves from supply constraints and pricing power.

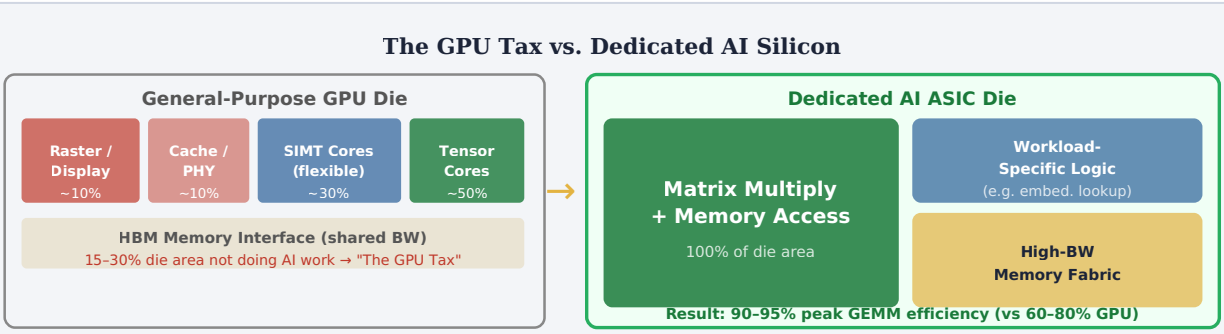


Figure 3A.1 — GPU die area allocation vs. dedicated AI ASIC. The "GPU Tax" is 15–30% of die area serving non-AI workloads.

3A.2 Google TPU — The Systolic Array Accelerator

3A.2.1 Architecture: Systolic Arrays vs SIMT

The TPU's core compute primitive is a **systolic array** — a fundamentally different architectural paradigm from GPU SIMT. In a systolic array, data flows rhythmically through a grid of Processing Elements (PEs), each multiplying one input and passing the result to its neighbor. Weights are loaded *once* into the array and stay stationary while activations stream through.

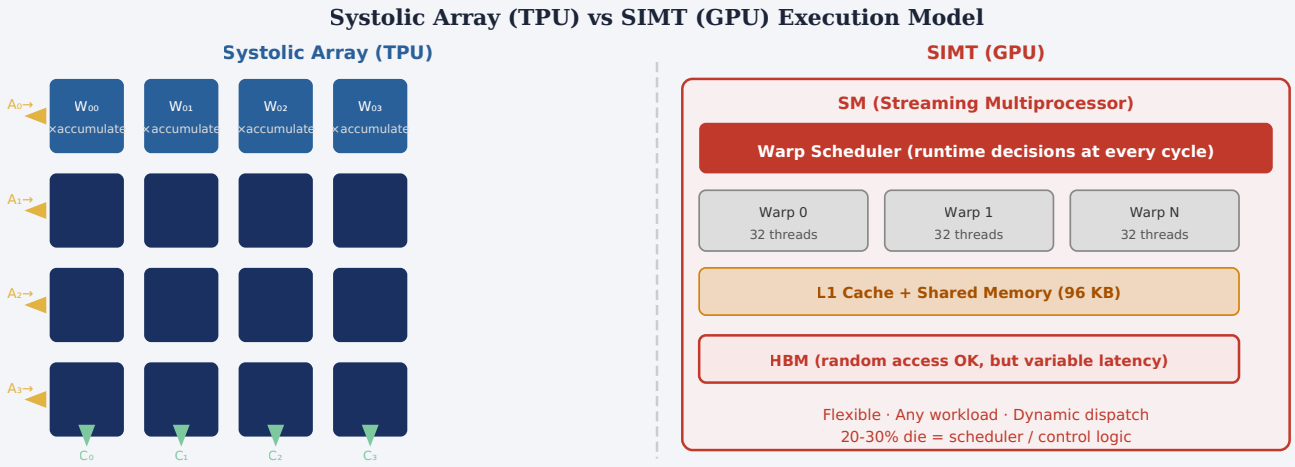


Figure 3A.2 — Systolic array (TPU) vs SIMT (GPU) execution model. TPU weights are stationary; activations and partial sums flow through the array. GPU warps are dynamically scheduled with full random memory access.

Property	Systolic Array (TPU)	SIMT (GPU)
Control model	No control flow	Arbitrary branches
Memory access	Stationary, rhythmic	Random access OK
Flexibility	GEMM-optimized	Any parallel op
Control logic area	**~5% of die**	**~20–30% of die**
Peak GEMM efficiency	**90–95%**	**60–80% typical**
Irregular workloads	**Poor (dynamic shapes)**	**Good**
Programming model	XLA / JAX (compiler)	CUDA / Triton (explicit)

3A.2.2 TPU Generation Progression

```
TPU GENERATION PROGRESSION:

TPU v1 (2016) — Inference-only:   [SHIPPED]
  Matrix multiplier: 256×256 = 65,536 MACs/cycle
  Memory: 28 MB on-chip SRAM, no HBM
  Purpose: Serving RankBrain; not for training

TPU v4 (2021–2023) — Dominant Google AI chip:  [SHIPPED]
  Peak BF16: ~275 TFLOPS per chip
  HBM: 32 GB HBM2, 1.2 TB/s bandwidth
  ICI: 800 Gbps per port, 6 ports = 4.8 Tbps per chip
  Pods: 4,096 chips via ICI in 3D torus topology
  Power: ~170W (vs H100's 700W)

TPU v5e (2023) — Inference-optimized:  [SHIPPED]
  Peak INT8: 393 TOPS
  Price: ~$0.35/hr on GCP (vs H100 ~$2.50/hr equivalent)

TPU v5p (2024) — Training powerhouse:  [SHIPPED]
  HBM: 95 GB per chip, 2.76 TB/s bandwidth
  ICI: 4,800 Gbps per chip
  Used for: Gemini Ultra training

Trillium (TPU v6, 2024–2025):  [SHIPPING]
  4.7× peak compute vs v5e
  HBM: 192 GB per chip  ← matches AMD MI300X!
  SparseCore: dedicated sparse embedding accelerator (new)
  Used for: Gemini 1.5 Pro/Ultra serving
```

3A.2.3 TPU Pods — The Interconnect Advantage

The most distinctive feature of TPU systems is their interconnect topology. Where GPU clusters use InfiniBand or RoCE with external switches, TPUs use **ICI (Inter-Chip Interconnect)** — a direct chip-to-chip fabric without network switches. This creates a fundamentally different communication model at scale.

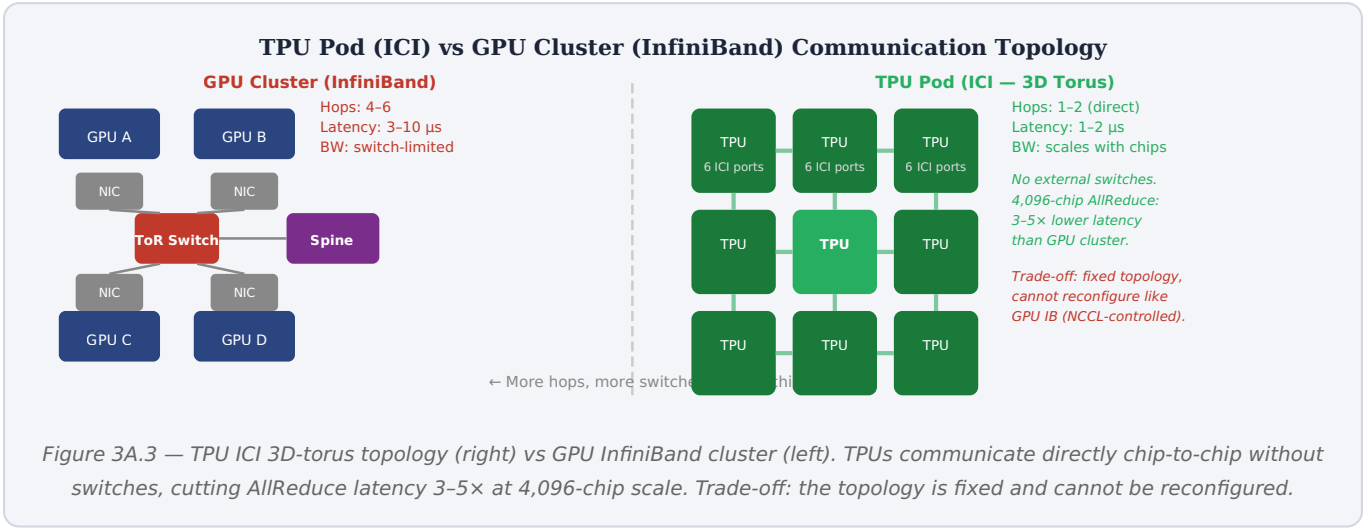


Figure 3A.3 — TPU ICI 3D-torus topology (right) vs GPU InfiniBand cluster (left). TPUs communicate directly chip-to-chip without switches, cutting AllReduce latency 3–5× at 4,096-chip scale. Trade-off: the topology is fixed and cannot be reconfigured.

3A.2.4 XLA Programming Model — Strength and Constraint

```
WHAT XLA PROVIDES:
  ✓ Automatic operator fusion (described in Ch 9)
  ✓ Static shape compilation — all shapes known at compile time
  ✓ Optimal tiling for systolic array dimensions
  ✓ Automatic replication/partitioning across TPU pods
  ✓ Guaranteed execution plan — no runtime dynamic dispatch

WHAT XLA RESTRICTS:
  ✗ No dynamic shapes (variable sequence lengths → bucketing or padding required)
  ✗ No custom kernels without XLA custom_call (complex API)
  ✗ No PyTorch eager mode on TPU
  ✗ Long compilation times: first run 5–30 minutes for large models
  ✗ Debugging is difficult (HLO IR is not human-readable)

  import jax, jax.numpy as jnp

  @jax.jit              # triggers XLA compilation on first call
  def transformer_layer(x, weights):
      return jnp.dot(x, weights['qkv'])  # compiled to systolic array MXU ops

  devices = jax.devices('tpu')          # returns list of TPU chips
  parallel_layer = jax.pmap(transformer_layer, axis_name='batch')  # auto DP
```

WHEN TO CHOOSE TPU VS GPU

Choose TPU when: using Google Cloud · primary framework is JAX · models have static shapes · scale >1,000 chips · training cost is primary concern. Choose GPU when: using PyTorch · variable sequence lengths · rapid prototyping · custom kernels needed · multi-cloud or on-premise deployment · quantization-heavy workloads (NVIDIA FP8/FP4 ahead).

3A.3 Apple Neural Engine — On-Device AI

3A.3.1 Architecture Overview

The Apple Neural Engine (ANE) is a dedicated AI accelerator embedded in every Apple Silicon chip (A-series and M-series). Unlike cloud accelerators optimized for throughput, the ANE is designed for **extreme energy efficiency** — running AI models on battery-powered devices. The key architectural differentiator is *unified memory*: CPU, GPU, and ANE share the same DRAM pool, eliminating the PCIe transfer overhead that burdens conventional GPU-attached systems.

```
APPLE NEURAL ENGINE EVOLUTION:

A11 (2017) — First ANE: 600 GOPS, 2-core, Face ID/Siri
A15 / M1 (2021): 15.8 TOPS INT8, 16-core ANE
A16 (2022): 17 TOPS, TSMC 4nm
A17 Pro / M3 (2023): 35 TOPS, TSMC 3nm, transformer optimizations
A18 / M4 (2024): [SHIPPED]
  38 TOPS, TSMC 3nm 2nd gen, native attention acceleration
  Use: On-device LLMs (Llama 3.2 3B/11B)
M4 Max (2024): [SHIPPED]
  38 TOPS ANE + 128 GB unified memory
  Runs: LLaMA-3 70B locally (quantized to INT4)
  Memory bandwidth: 546 GB/s unified (CPU/GPU/ANE share pool)

ON-DEVICE LLM PERFORMANCE (Apple M4 Max, 128 GB):
Model            Format    Speed         Notes
──────────────────────────────────────────────────
Llama 3.2 3B     INT4      ~150 tok/s    Runs on A18 iPhone too
Llama 3.2 11B    INT4      ~60 tok/s     Requires M1+ (16 GB min)
Llama 3 70B      INT4      ~18 tok/s     M4 Max / Pro only
Mistral 7B       INT4      ~90 tok/s     Excellent perf/watt
Gemma 2 27B      INT4      ~35 tok/s     Requires 32 GB+

PERFORMANCE-PER-WATT COMPARISON:
M4 Max ANE+GPU (70B INT4): ~18 tok/s at ~15W = 1.2 tok/s/W
H100 SXM       (70B INT4): ~3,500 tok/s at 700W = 5.0 tok/s/W

GPU wins per watt — BUT M4 Max serves ONE user on a laptop battery.
H100 serves THOUSANDS of users simultaneously.
These workloads are NOT comparable — different use-case regimes.
```

3A.4 AWS Trainium and Inferentia — Cloud Custom Silicon

3A.4.1 AWS Inferentia2 — Dedicated Inference ASIC

AWS Inferentia2 (2023) is Amazon's purpose-built inference accelerator. The economic case is compelling: at roughly half the cost per million tokens compared to A100 GPUs, Inferentia2 demonstrates the TCO advantage of dedicated silicon for stable, high-volume inference workloads.

```
AWS INFERENTIA2 ARCHITECTURE: [SHIPPED]
  Process: TSMC 7nm
  Compute: 2× NeuronCore v2 per chip
  Peak BF16: 380 TFLOPS per chip; FP8: 760 TFLOPS per chip
  Memory: 32 GB HBM2e per chip, 820 GB/s bandwidth
  Max instance: inf2.48xlarge — 12 chips, 384 GB total HBM

COST ANALYSIS — LLaMA-2 70B inference:
  inf2.48xlarge (12 Inferentia2): ~1,800 tok/s at $16.37/hr
  p4d.24xlarge  (8× A100):        ~2,100 tok/s at $32.77/hr

  Cost/1M tokens:
    Inferentia2: $16.37 / (1,800 × 3,600 / 1,000,000) = $2.53
    A100:        $32.77 / (2,100 × 3,600 / 1,000,000) = $4.33

  → Inferentia2 is 41% cheaper per token than A100.
    This is the economic case for dedicated inference silicon.

KEY LIMITATIONS:
  ✗ Neuron SDK supports limited ops (custom ops fall back to CPU)
  ✗ Dynamic shapes poorly supported (pad to fixed max length)
  ✗ Only available on AWS — no on-premise option
  ✗ Smaller ecosystem vs CUDA
```

3A.4.2 AWS Trainium2 — Training Accelerator

```
AWS TRAINIUM2 SPECIFICATIONS (2024): [SHIPPED]
  Chips per server: 16 Trainium2 (vs 8 GPUs in typical GPU server)
  Peak BF16: ~3.7 PFLOPS per chip
  Memory: 96 GB HBM per chip, 1.54 TB/s bandwidth
  Interconnect: NeuronLink 4 — 1,600 Gbps per chip
  Instance: trn2.48xlarge — 16 chips, 1.5 TB total HBM

TCO COMPARISON — Fine-tuning LLaMA-3 70B (1 epoch, 50B tokens):
  On H100:      ~480 GPU-hours  @ $2.00/GPU-hr  = ~$960
  On Trainium2: ~380 Trn2-hours @ $1.30/Trn2-hr = ~$494
  Savings: 49% cost reduction

NOTE: Results depend heavily on model architecture and Neuron SDK
optimization quality. NVIDIA's software advantages can partially close this gap.
```

3A.5 Intel Gaudi 3 — The Open Ecosystem Challenger

Intel Gaudi 3 (launched 2024) offers a compelling value proposition for cost-sensitive training: ~4× cheaper server hardware than H100, integrated networking, and open Ethernet connectivity — at the cost of being 1.87× slower on equivalent workloads (MLPerf v4.1).

```
INTEL GAUDI 3 SPECIFICATIONS: [SHIPPED]
  Process: TSMC 5nm
  Compute: ~800 TFLOPS BF16 (actual comparable, vs 1,835 claimed in marketing)
  Memory: 128 GB HBM2e, 3.7 TB/s bandwidth
  Network: 24× 200GbE RoCE ports — INTEGRATED ON-DIE (unique!)
  Power: ~900W TDP
  Server price: ~$65,000 (vs $250,000+ for 8× H100)

KEY DIFFERENTIATOR — Integrated RDMA Networking:
  24 × 200 Gbps = 4.8 Tbps network ON-DIE
  H100 requires 8 separate InfiniBand NICs + PCIe switch → adds cost + power
  Gaudi 3 server is ~15% cheaper to build (NIC cost eliminated)

PERFORMANCE vs H100 (MLPerf v4.1):
  LLaMA-2 70B training to accuracy target:
    8× H100:    16.8 minutes
    8× Gaudi 3: 31.5 minutes  (1.87× slower)

  Break-even: Gaudi 3 cheaper at <60% utilization; H100 better at >80%

ECOSYSTEM STATUS (2026):
  ✓ PyTorch, DeepSpeed, HuggingFace Optimum Habana
  ✗ No Triton backend · No FlashAttention v3 · No CUDA Graph equivalent
```

3A.6 Meta MTIA v2 — Custom Silicon for Recommendation

3A.6.1 Why Recommendation Models Need Custom Silicon

Meta's largest AI workload is NOT LLM inference — it is recommendation and ranking for Facebook, Instagram, and WhatsApp ads. These models have fundamentally different characteristics: sparse embedding lookups that access thousands of random addresses per inference. GPU HBM is optimized for sequential coalesced access — random access collapses effective bandwidth by 6–10×.

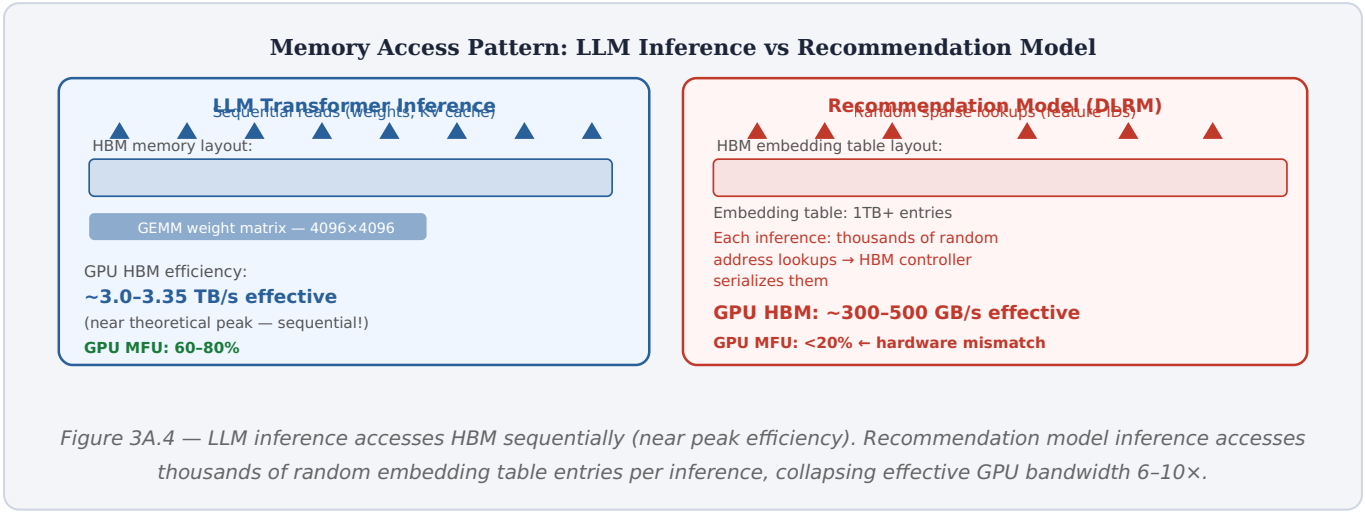


Figure 3A.4 — LLM inference accesses HBM sequentially (near peak efficiency). Recommendation model inference accesses thousands of random embedding table entries per inference, collapsing effective GPU bandwidth 6–10×.

```
MTIA v2 SPECIFICATIONS (2024): [SHIPPED]
  Process: TSMC 5nm
  Compute: ~192 TFLOPS BF16 (low — compute is NOT the bottleneck)
  Memory: 64 GB LPDDR5 + on-chip SRAM
  Memory bandwidth: 1.7 TB/s (LPDDR5 optimized for random access)
  Power: ~90W (dramatically lower than GPU)
  Specialized units: Embedding lookup engine · Small GEMM engine

KEY ARCHITECTURAL CHOICE — LPDDR5 over HBM:
  HBM:    3.35–8 TB/s peak sequential · terrible random access latency
  LPDDR5: ~1.7 TB/s peak sequential · MUCH better random access
           (many independent banks, lower queue depth)

  For embedding lookup (random access):
    GPU HBM effective BW:     ~300–500 GB/s  (random access penalty)
    MTIA LPDDR5 effective BW: ~900–1,200 GB/s
    MTIA is 2–3× faster at embedding lookup despite lower peak BW!

AT META SCALE:
  Deployed: Tens of thousands of MTIA v2 chips
  Inference volume: "Hundreds of trillions per day"
  Cost vs GPU: ~40–50% lower TCO for recommendation workloads
```

3A.7 Groq LPU — Deterministic Latency

3A.7.1 The Static Dataflow Architecture

Groq's Language Processing Unit takes a completely different approach: instead of maximizing throughput, it maximizes **deterministic, predictable latency**. The key insight is that GPU latency variance stems from dynamic scheduling — warp schedulers making runtime decisions based on cache state and memory bank conflicts. Groq eliminates runtime decisions entirely.

GROQ LPU CORE DESIGN:

Problem with GPU inference:
  Dynamic scheduling → latency varies based on memory bank conflicts, cache behavior
  P99 latency >> P50 latency → hard to guarantee SLAs for interactive AI

Groq's solution: STATIC DATAFLOW ARCHITECTURE
  ALL computation scheduled at COMPILE TIME — no runtime decisions
  Compiler determines exactly:
    - When each instruction executes (to the cycle)
    - Where each piece of data lives at each cycle
    - Which memory bank is accessed at each step

Result:
  P50 latency == P99 latency (zero variance)
  No cache misses (no caches — all memory access deterministic)
  No warp scheduling overhead

GROQ CHIP SPECIFICATIONS (GroqChip 1): **[SHIPPED]**
  SRAM: 230 MB on-chip (massive — replaces HBM for small models)
  Compute: ~750 TOPS INT8
  Power: ~300W
  Key: NO HBM — entire working set in on-chip SRAM

SRAM vs HBM for working set that fits:
  SRAM bandwidth: ~80 TB/s
  H100 HBM bandwidth: ~3.35 TB/s
  → 24× faster memory access for the active working set!

GROQ PERFORMANCE (2024):
  LLaMA-3 70B decode latency: ~17 ms/token (P99 ≈ P50)
  Comparable H100:   ~25–40 ms/token (with variance)
  Advantage: 1.5–2.4× lower latency + zero variance
  Trade-off: Lower throughput per dollar vs GPU for batched workloads
  Use case: Ultra-low-latency interactive applications (code completion, voice AI)

3A.7.2 GroqRack Scale

```
GROQRACK ARCHITECTURE:
  9 GroqCards per rack
  Each GroqCard: 8 GroqChips
  Total: 72 GroqChips per rack
  Aggregate SRAM: 72 × 230 MB = 16.6 GB (ultra-fast)
  External DRAM (larger models): 18 TB/s aggregate bandwidth

  LLaMA-3 405B on GroqRack:
    Weights distributed across 72 chips' DRAM
    Streamed to SRAM in pipeline
    Execution: fully deterministic end-to-end
    Throughput: lower than H100 cluster; Latency: 2–4× lower
```

3A.8 Tenstorrent — The Open RISC-V Bet

Tenstorrent (founded by Jim Keller, former Apple Silicon and AMD Zen architect) takes a long-term open-hardware bet: building AI accelerators on RISC-V cores, eliminating instruction set licensing, and enabling fully open compiler stacks. Each "Tensix core" is a small RISC-V processor paired with a matrix engine and SRAM tile — programmable in a way that TPU systolic arrays are not.

```
TENSTORRENT BLACKHOLE (2024): [SHIPPED — limited]
  140 Tensix cores per chip (each = RISC-V + matrix engine + 1.5 MB SRAM)
  Peak: ~735 TFLOPS BF16
  Memory: 32 GB LPDDR5, 768 GB/s
  On-chip SRAM: 210 MB total

STATUS (2026):
  ✓ Growing interest from research labs and edge AI
  ✓ Open-source compiler potential (MLIR backend in development)
  ✗ Not yet competitive with H100/MI300X for large-scale training
  ✗ Ecosystem thin vs CUDA

Long-term potential: open AI hardware ecosystem play as RISC-V matures
```

3A.9 The Accelerator Selection Framework

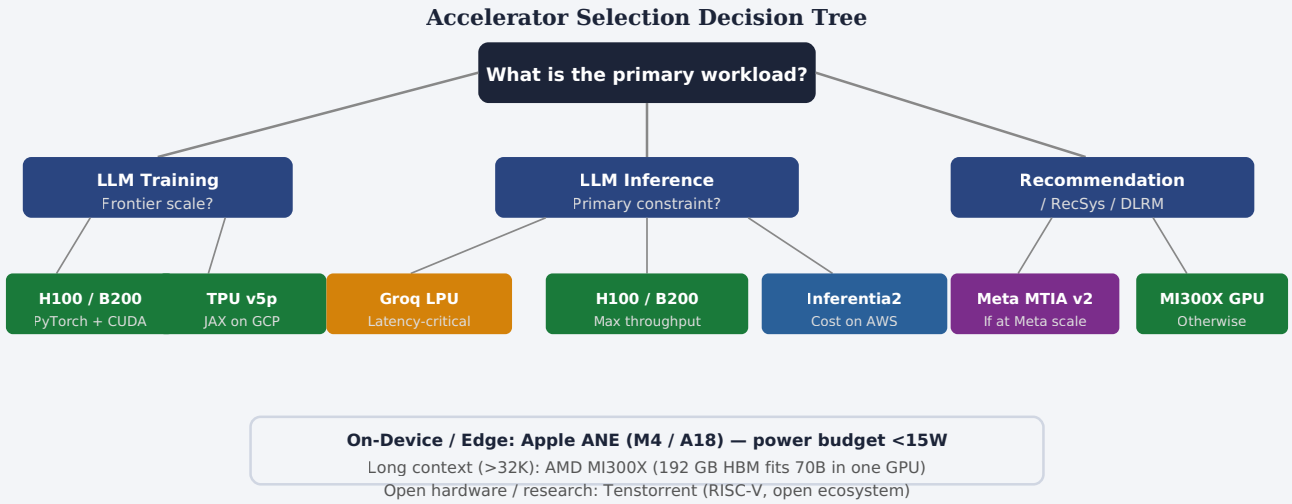


Figure 3A.5 — Accelerator selection decision tree. The primary workload characteristic — training vs. inference × throughput vs. latency vs. cost — determines the optimal choice.

Workload	Best Choice	Why	Avoid
LLM Training (frontier)	**H100 / B200**	CUDA ecosystem, FP8	TPU (JAX friction)
LLM Training (JAX)	**TPU v5p**	ICI fabric, native JAX	GPU (Python overhead)
LLM Inference (latency)	**Groq LPU**	Deterministic P99	GPU (latency variance)
LLM Inference (throughput)	**H100 / B200**	Best tok/s/$ overall	TPU v5e (limited flex)
LLM Inference (cost, AWS)	**Inferentia2 / TPU v5e**	41% cheaper than A100	On-demand GPU
Long context (>32K)	**AMD MI300X**	192 GB HBM in one GPU	TPU (shape constraints)
Recommendation / DLRM	**MTIA / MI300X**	Random access BW	TPU (poor random access)
On-device inference	**Apple ANE**	Battery efficiency	GPU (power too high)
Cost-sensitive training	**Gaudi 3 / Trainium2**	4× cheaper hardware	H100 (premium price)
Research / flexibility	**GPU (any)**	PyTorch ecosystem	All others (friction)

3A.9.2 Performance-Per-Dollar Comparison

```
COST EFFICIENCY — LLM Inference, LLaMA-3 70B (April 2026 cloud rates):

                      $/hr     Tok/s    $/1M tokens   vs H100
──────────────────────────────────────────────────────────────
H100 SXM (8× GPU)    ~$16     ~9,000   $0.49         Baseline
H200 SXM (8× GPU)    ~$22     ~13,000  $0.47         -4%
B200 (8× GPU)        ~$40     ~30,000  $0.37         -24%
TPU v5e (8× chip)    ~$13     ~7,200   $0.50         +2%
AWS Inferentia2      ~$16     ~7,500   $0.59         +20%
MI300X (8× GPU)      ~$20     ~9,200   $0.60         +22%
Groq (per token)     $0.27/1M ~varies  $0.27          -45%

Notes:
  Groq prices per-token, not per-hour — latency advantage not shown
  MI300X advantage for 70B: fits in 1 GPU vs 2× H100 for FP16
  On-premise TCO changes math significantly (see Chapter 5)
```

3A.9.3 Build vs Buy — Custom Silicon Decision Framework

```
CUSTOM SILICON DECISION FRAMEWORK:

1. SCALE: >$500M/year inference spend? → custom silicon may be justified
          Below this: buy commercial, optimize in software

2. WORKLOAD UNIQUENESS: Recommendation models, unusual sparsity, MoE routing
   → may justify custom silicon. Standard transformer → H100/B200/TPU already optimal

3. ENGINEERING DEPTH: $50–500M NRE, 3–5 year timeline for custom chip
   Chiplet customization: $5–20M, 1–2 years
   FPGA prototyping: $1–5M, months

4. RISK TOLERANCE: single tape-out failure = $5–50M wasted, 18-month delay

REAL-WORLD THRESHOLDS (2026):
  Meta (MTIA):   ~$10B/year inference spend → clearly justified
  Apple (ANE):   billions of devices × marginal cost → clearly justified
  AWS (Inf/Trn): cloud revenue leverage → justified
  $20M/year startup: NOT justified → use commercial hardware
```

3A.10 The Principal Architect's View — Heterogeneous Infrastructure

Production AI infrastructure increasingly mixes accelerator types. The optimal architecture routes workloads to hardware based on their characteristics — not a single hardware platform for everything.

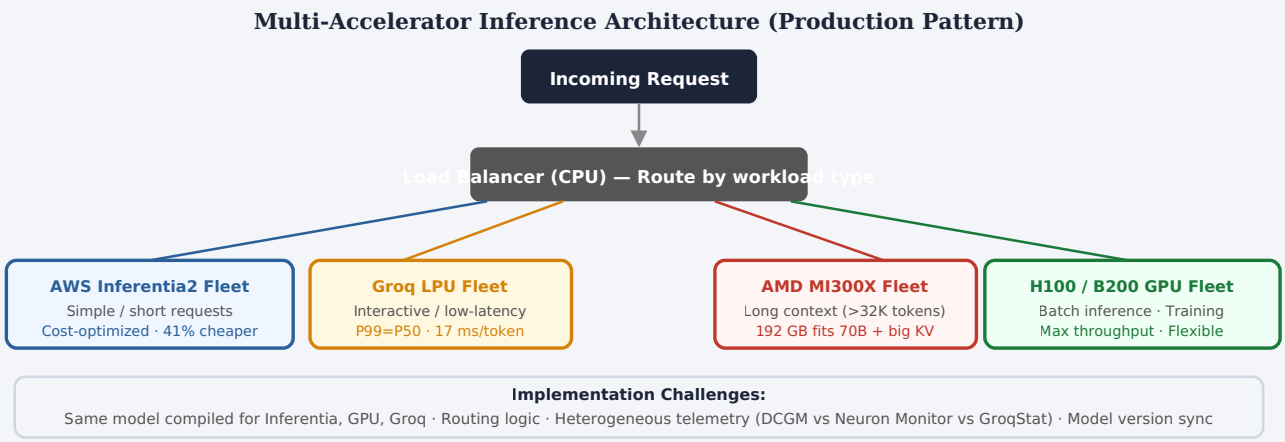


Figure 3A.6 — Multi-accelerator production inference architecture. Workloads are routed by type: cost-optimized requests to Inferentia, latency-critical to Groq, long-context to MI300X, high-throughput to H100/B200.

Key Numbers to Know Cold

GOOGLE TPU TRILLIUM (V6)

~4.7× v5e compute · 192 GB HBM · ICI fabric

Best for: JAX training · Google Cloud · static-shape workloads · large-scale Google model training

APPLE M4 MAX ANE

38 TOPS · 128 GB unified · ~15W

Best for: On-device inference · privacy-critical applications · single-user 70B INT4 locally

AWS INFERENTIA2

380 TFLOPS BF16 · 32 GB · $16.37/hr

Best for: Cost-optimized AWS inference (41% cheaper than A100 per million tokens)

INTEL GAUDI 3

~800 TFLOPS BF16 · 128 GB · 24×200GbE

Best for: Cost-sensitive training · open networking · 4× cheaper server vs H100

META MTIA V2

192 TFLOPS BF16 · 64 GB LPDDR5 · 90W

Best for: Recommendation/ranking · random memory access · hyperscale embedding workloads

GROQ LPU

~750 TOPS INT8 · 230 MB SRAM · 300W

Best for: Ultra-low deterministic latency · <17 ms/token P99 · interactive AI SLA guarantee

AMD MI300X CORRECTION (COMMON MISCONCEPTION)

MI300X architecture: 6 XCDs (Compute Dies) + 4 MCDs (Memory Controller Dies) = 10 total chiplets. 304 Compute Units total. Not "3 chiplets" — that is the most common misconception about this architecture.

Chapter 3A Review Questions

1. Explain the systolic array architecture used in Google TPUs. Why does it achieve higher efficiency for GEMM than NVIDIA's SIMT model? What workloads does it handle poorly that GPUs handle well? Draw the data flow diagram from memory.
2. A startup is deciding between deploying on AWS Inferentia2 vs H100 GPUs for serving LLaMA-3 70B at 10,000 requests/day with average output length of 500 tokens. Calculate approximate monthly cost for each option using the formulas in this chapter. Which is cheaper and under what assumptions does that change?
3. Why does Meta's MTIA v2 use LPDDR5 instead of HBM? Explain the memory access pattern of recommendation model inference and why effective bandwidth differs so dramatically from peak bandwidth on GPU HBM.
4. Groq claims "P99 latency = P50 latency" for LLM inference. Explain the architectural mechanism that makes this possible. What is the fundamental trade-off Groq makes to achieve this property, and for which workloads does this trade-off make economic sense?
5. A company runs 50% of its inference on LLMs (70B class) and 50% on recommendation models (DLRM-style). They currently use all H100 GPUs. Design a heterogeneous hardware strategy, estimate the cost reduction, and identify the top three implementation challenges in deploying it.
