End-to-End Machine Learning Systems

Lecture 2: Types of ML Systems

Prof. Tanu Malik, University of Missouri

Acknowledgements: These lecture materials are adapted from the original notes provided by Prof. Vijaya Reddi and Chip Huyen. Any deviations, errors, or supplementary content in these slides are solely my responsibility.

Three Analytical Frameworks

The three analytical frameworks:
1. **D-A-M:** Data, Algorithm, Machine — where is the bottleneck?
2. **Iron Law:** Latency = Work / Machine capacity — how long will it take?
3. **Degradation Equation:** Accuracy = f(Data, Model, Time) — when will it fail?

The Data-Algorithms-Machine (DAM) Taxonomy

Every ML system has three interdependent axes:

Data — D_{vol}, volume, quality, distribution
How many bytes must move?

Algorithm — O, architecture, operation count
How many FLOPs, what parallelism pattern?

Machine — BW, R_{peak}, η, L_{lat}
How fast can this silicon *move* bytes and *do* operations — and what fixed tax per task?

Bandwidth is a property of the machine, not of the data.

Axis	**Variable**	**Unit**
Data	D_{vol}	GB
Algorithm	O	FLOPs
Machine	BW	GB/s
	R_{peak}	FLOP/s
	η	—
	L_{lat}	s

A Neural Network

A NN:

$$y = Wx \qquad \text{(then } y \text{ becomes the input of the next layer)}$$

Weights W

Learned during training
Frozen during inference
Identical for every input

ResNet-50: 25.6M weights $\rightarrow$ 97.5 MB at FP32

Activations x, y

Computed fresh per input
Discarded afterwards
Different for every input

ResNet-50: $\approx$ 20 MB of activations **per image** at FP32[a]

[a] All layers, summed

What is the memory use if a the network is input a batch of **256 images**?

A Neural Network

A NN:

$$y = Wx \qquad \text{(then } y \text{ becomes the input of the next layer)}$$

Weights W

Learned during training
Frozen during inference
Identical for every input

ResNet-50: 25.6M weights $\rightarrow$ 97.5 MB at FP32

Activations x, y

Computed fresh per input
Discarded afterwards
Different for every input

ResNet-50: $\approx$ 20 MB of activations **per image** at FP32[a]

[a] All layers, summed

What is the memory use if a the network is input a batch of **256 images**?

W is loaded **once**. Activations are produced **256 times**.

Weights are a fixed cost per batch. Activations are a per-image cost.

D_{vol} is Bytes *Moved*, Not Bytes *Stored*

Quantity	**Example**	**Where it lives**
Dataset size	100 images = 60 MB	at rest, on disk
Model size	25.6M params = 97.5 MB	at rest, in a file
D_{vol}	**see below**	**in flight, memory $\leftrightarrow$ processor**

Classifying 100 images on ResNet-50:

	Bytes	**Share**
Images	60 MB	3%
Weights (loaded *once*)	97.5 MB	5%
Activations (100×20 MB)	2,000 MB	**92%**
D_{vol}	2,158 MB	

$$D_{\text{vol}} = \underbrace{W}_{\text{fixed per batch}} + B \cdot \underbrace{A}_{\text{per image}}$$

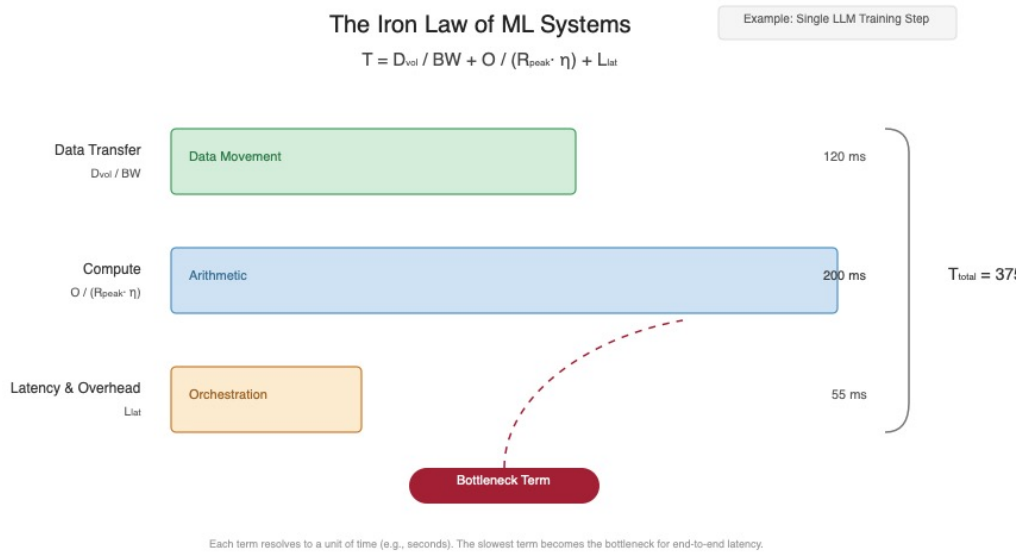
Training vs. Inference Costs

	Inference	**Training**
Direction	forward only	forward + backward
Weights	read-only	read **and** written
Gradients	—	one per weight ($1\times$ model)
Optimizer state	—	Adam keeps $2\times$ model
Activations	discard after each layer	**retained** for the backward pass
Memory for state	$1\times$ model	$\approx 4\times$ model
Work per input	≈ 2 FLOPs/param	≈ 6 FLOPs/param

The backward pass costs roughly *twice* the forward pass, and it forces to keep everything the forward pass produced. Hence $2 \to 6$ FLOPs per parameter, and $1\times \to 4\times$ the state.

This is why your phone can run a model it could never train.

Iron Law: Three Terms



Data $\frac{D_{\text{vol}}}{BW}$

Bytes / (Bytes/s) = **s**

Compute $\frac{O}{R_p \cdot \eta}$

FLOPs / (FLOPs/s) = **s**

Overhead L_{lat}

Orchestration tax = **s**

$$T = \overbrace{\frac{D_{\text{vol}}}{\underbrace{BW}_{\text{machine}}}}^{\text{data}} + \overbrace{\frac{O}{\underbrace{R_{\text{peak}} \cdot \eta}_{\text{machine}}}}^{\text{algorithm}} + \underbrace{L_{\text{lat}}}_{\text{machine}}$$

For **one task**, the terms **add**: the largest term sets the floor, the rest are added.

What about the network?

Our convention: L_{lat} is *on-machine* fixed overhead paid per task If the task is **offloaded**, Iron Law includes a fourth term $T_{\text{net}} = \frac{D_{\text{vol}}}{BW_{\text{net}}} + L_{\text{net}}$.

For an offloaded task, the Iron Law is:

$$T = \frac{D_{\text{vol}}}{BW} + \frac{O}{R_{\text{peak}} \cdot \eta} + L_{\text{lat}} + \frac{D_{\text{net}}}{BW_{\text{net}}} + L_{\text{net}}$$

1. D_{net} are bytes on the wire and D_{vol} are bytes crossing the memory/processor boundary at whichever machine runs the task.
2. Typical bandwidth rates differ signficantly: BW_{net} is often 1-3 TB/s for HBM, while BW_{net} is often 5-10 GB/s for a 10 Gbps network.

Iron Law: An Example

Training one batch of GPT-3 (175B parameters) *[one task → additive form]*

Term	Variables	Calculation	Time
Data	D_{vol}=350 GB, BW=3,350 GB/s	350/3,350	**0.10 s**
Compute	O=3.14×10^{14}, R_p=989 TF, η=0.5	$3.14 \times 10^{14} / (989 \times 10^{12} \times 0.5)$	**0.64 s**
Overhead	L_{lat} (sync, sched.)	—	**0.05 s**
		$T_{\text{total}} \approx$	**0.79 s**

$D_{\text{vol}} = 350$ GB is 175B parameters at FP16 — *weights alone*. Gradients and optimizer state make real training state several times larger, so this is a floor.

Diagnosis:

Iron Law: An Example

Training one batch of GPT-3 (175B parameters) *[one task → additive form]*

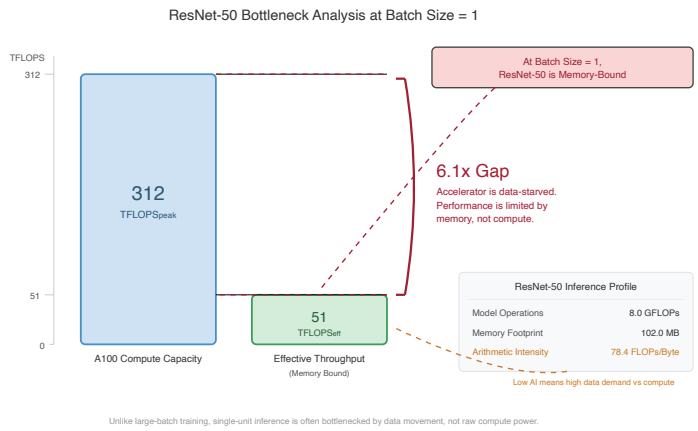
Term	Variables	Calculation	Time
Data	D_{vol}=350 GB, BW=3,350 GB/s	350/3,350	**0.10 s**
Compute	O=3.14×10^{14}, R_p=989 TF, η=0.5	$3.14 \times 10^{14} / (989 \times 10^{12} \times 0.5)$	**0.64 s**
Overhead	L_{lat} (sync, sched.)	—	**0.05 s**
		$T_{\text{total}} \approx$	**0.79 s**

$D_{\text{vol}} = 350$ GB is 175B parameters at FP16 — *weights alone*. Gradients and optimizer state make real training state several times larger, so this is a floor.

Diagnosis:

1. The compute term dominates for LLM training — by more than 6×.
2. Upgrading memory bandwidth alone yields almost nothing.

Iron Law: ResNet-50 — A Different Bottleneck



ResNet-50 inference (batch = 1) *[one task]*

Quantity	**Value**
Operations O	7.7 GFLOP
Weights W	97.5 MB
Activations $1 \times A$	20 MB
D_{vol}	**117.5 MB**
Data term D_{vol}/BW	**0.058 ms**
Compute term O/R_p	**0.025 ms**

The accelerator finishes its arithmetic in half the time it takes to feed it. It computes, then *waits*.

Data term dominates — the opposite of GPT-3 training.

Diagnose the Bottleneck

Diagnose the Bottleneck *[one task]*

A mobile inference pipeline has:

- $D_{\text{vol}} = 40$ MB, $BW = 100$ MB/s
- $O = 2 \times 10^9$ FLOPs, $R_{\text{peak}} = 20$ GF, $\eta = 0.6$
- $L_{\text{lat}} = 5$ ms

Calculate T_{total} and identify the dominant term.

Diagnose the Bottleneck

Diagnose the Bottleneck *[one task]*

A mobile inference pipeline has:

- $D_{\text{vol}} = 40$ MB, $BW = 100$ MB/s
- $O = 2 \times 10^9$ FLOPs, $R_{\text{peak}} = 20$ GF, $\eta = 0.6$
- $L_{\text{lat}} = 5$ ms

Calculate T_{total} and identify the dominant term.

Keep these three numbers on the board. We are going to ask a different question about the same system.

Solution:

Data: $40/100 = 400$ ms

Compute: $\frac{2 \times 10^9}{12 \times 10^9} = 167$ ms

Overhead: 5 ms

$T_{\text{total}} = \mathbf{572}$ **ms**

Data-bound!

Which Term Is Bigger?

When is the compute term bigger than the data term?

$$\frac{O}{R_p \cdot \eta} > \frac{D_{\text{vol}}}{BW}$$

Which Term Is Bigger?

When is the compute term bigger than the data term?

$$\frac{O}{R_p \cdot \eta} > \frac{D_{\text{vol}}}{BW}$$

$$\frac{O}{R_p \cdot \eta} > \frac{D_{\text{vol}}}{BW} \iff \underbrace{\frac{O}{D_{\text{vol}}}}_{\textbf{workload}} > \underbrace{\frac{R_p \cdot \eta}{BW}}_{\textbf{machine}}$$

Which Term Is Bigger?

When is the compute term bigger than the data term?

$$\frac{O}{R_p \cdot \eta} > \frac{D_{\text{vol}}}{BW}$$

$$\frac{O}{R_p \cdot \eta} > \frac{D_{\text{vol}}}{BW} \iff \underbrace{\frac{O}{D_{\text{vol}}}}_{\textbf{workload}} > \underbrace{\frac{R_p \cdot \eta}{BW}}_{\textbf{machine}}$$

What the workload supplies

FLOPs per byte it touches

arithmetic intensity

What the machine demands

FLOPs per byte it can fetch

machine balance point

Which Term Is Bigger?

When is the compute term bigger than the data term?

$$\frac{O}{R_p \cdot \eta} > \frac{D_{\text{vol}}}{BW}$$

$$\frac{O}{R_p \cdot \eta} > \frac{D_{\text{vol}}}{BW} \iff \underbrace{\frac{O}{D_{\text{vol}}}}_{\textbf{workload}} > \underbrace{\frac{R_p \cdot \eta}{BW}}_{\textbf{machine}}$$

What the workload supplies

FLOPs per byte it touches

arithmetic intensity

What the machine demands

FLOPs per byte it can fetch

machine balance point

	workload	**machine**	
ResNet-50, batch 1	66 FLOPs/B	153	starves → **data-bound**
GPT-3 training	897 FLOPs/B	148	fed → **compute-bound**

The Bottleneck Is Not a Property of the Model Alone

Coinsider ResNet-50 with one GPU and one precision. But change the batch size.

Batch	O **(GFLOP)**	D_{vol} **(MB)**	O/D_{vol}	**vs. machine (153)**
1	7.7	117.5	**66**	starves $\rightarrow$ **data-bound**
8	61.6	257.5	**239**	fed $\rightarrow$ **compute-bound**
64	493	1,377	**358**	compute-bound
256	1,971	5,218	**378**	compute-bound
∞	—	—	**385**	*ceiling* $= O_{img}/A_{img}$

1. Batching amortizes the *fixed* weight cost — and once activations dwarf the weights there is nothing left to amortize.
2. The ceiling 385 is set by the **architecture**, not the batch.
3. In Resnet-50, the regime flips from data-bound to compute-bound at $B \approx 3$, and saturates at $B \approx 64$.

Bottleneck Is Not a Property of the Model Alone (cont.)

"Is ResNet-50 compute-bound or data-bound?" **is a malformed question.**

The bottleneck is a property of **architecture** $\times$ **execution parameters** $\times$ **hardware** — *not* of the model alone.

One Task, or a Stream of Tasks?

Alice asks:

"A request arrives, the queue is empty. How long until the user sees a response?"

Latency question

Bob asks:

"We are fully loaded, requests arriving back to back. How many per second can we sustain?"

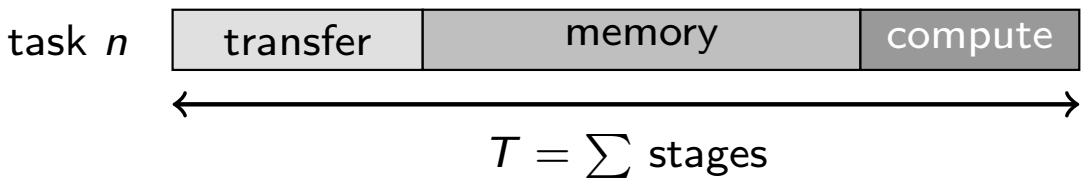
Throughput question

Same service. Same hardware. Same model.

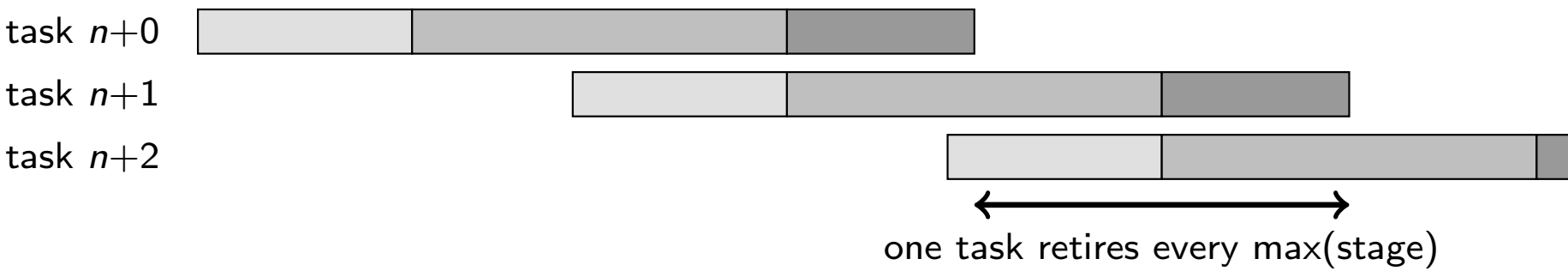
Same latency answer?

A stream has pipeline execution

One task: stages are serialized $\rightarrow$ they **add**



Stream of tasks: stages overlap $\rightarrow$ the widest one sets the rate



Iron Law for a Stream of Tasks

$$T_{\text{stream}} \geq \max\left(\underbrace{\frac{D_{\text{vol}}}{BW}}_{\text{data}}, \underbrace{\frac{O}{R_p \cdot \eta}}_{\text{compute}}, \underbrace{T_{\text{net}}}_{\text{offload, if any}}\right) + L_{\text{lat}}$$

- **Same stages as the Iron Law.** The max appears because now the stages are overlapped, not replaced.
- L_{lat} **stays outside the max.** Per-task fixed overhead does not overlap with itself; every task pays its own.
- **Why $\geq$, not $=$?**

Iron Law for a Stream of Tasks

$$T_{\text{stream}} \geq \max\left(\underbrace{\frac{D_{\text{vol}}}{BW}}_{\text{data}}, \underbrace{\frac{O}{R_p \cdot \eta}}_{\text{compute}}, \underbrace{T_{\text{net}}}_{\text{offload, if any}}\right) + L_{\text{lat}}$$

- **Same stages as the Iron Law.** The max appears because now the stages are overlapped, not replaced.
- L_{lat} **stays outside the max.** Per-task fixed overhead does not overlap with itself; every task pays its own.
- **Why $\geq$, not $=$?** Perfect overlap is an idealization. Real systems sit **between the sum and the max**, moving toward the max as prefetching and batching improve.

Batch Vs Pipeline Execution for the Mobile Pipeline

The mobile pipeline: data 400 ms · compute 167 ms · overhead 5 ms

Alice: one task

$400 + 167 + 5$

572 ms

latency of one request

Bob: stream of tasks

$\max(400, 167) + 5$

405 ms $\rightarrow$ 2.5 req/s

steady-state service time

Batch Vs Pipeline Execution for the Mobile Pipeline

The mobile pipeline: data 400 ms · compute 167 ms · overhead 5 ms

Alice: one task

$400 + 167 + 5$

572 ms

latency of one request

Bob: stream of tasks

$\max(400, 167) + 5$

405 ms $\rightarrow$ 2.5 req/s

steady-state service time

Now buy a GPU that is 2$\times$ faster. Compute:

Batch Vs Pipeline Execution for the Mobile Pipeline

The mobile pipeline: data 400 ms · compute 167 ms · overhead 5 ms

Alice: one task

$400 + 167 + 5$

572 ms

latency of one request

Bob: stream of tasks

$\max(400, 167) + 5$

405 ms $\rightarrow$ 2.5 req/s

steady-state service time

Now buy a GPU that is 2$\times$ faster. Compute: 167 $\rightarrow$ 83.5 ms.

572 $\rightarrow$ 488.5 ms $=$ **15% faster**

405 $\rightarrow$ 405 ms $=$ **0% faster**

The Rule of Thumb

Latency question $\rightarrow$ add the terms.

Throughput question $\rightarrow$ take the max.

Sum answers "how long for this one request?"
Max answers "how many per second?"

Compute SUM for the following terms

p99 · tail latency · deadline · SLA
time-to-first-token · "does it fit in X ms"

Compute MAX for the following terms

QPS · images/s · tokens/s
utilization · saturation · "at full load"

Ask which question you are answering **before** *you compute anything.*

Machine comparison for Training and Inference

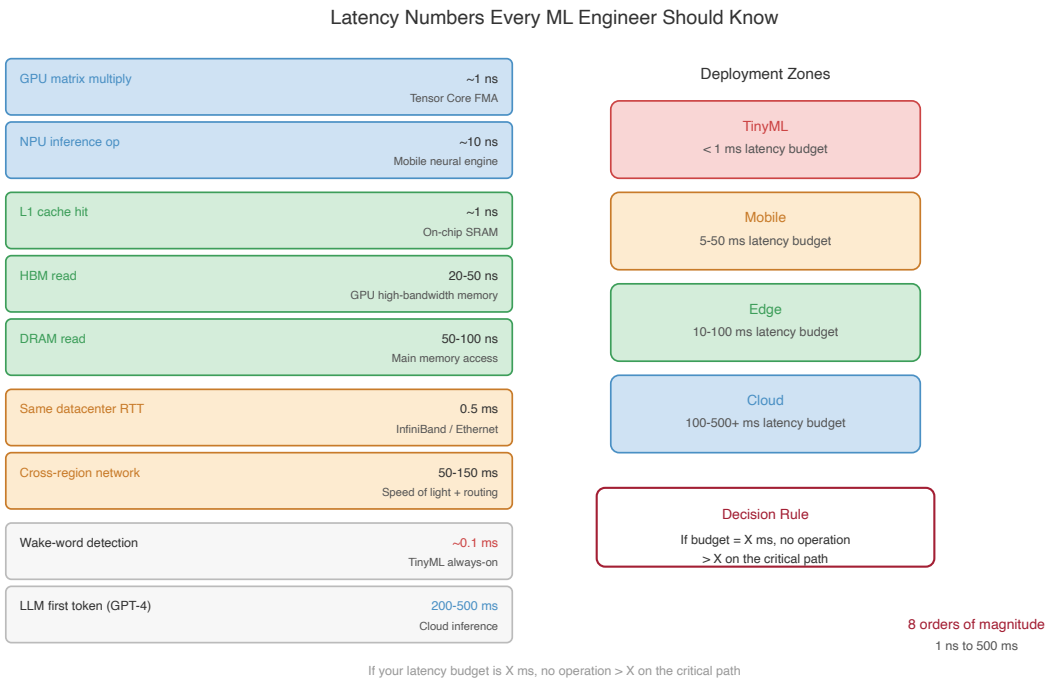
Tag	**Card**	R_p **(FP16)**	BW	**Balance @ peak**	**@** $\eta = 0.5$
[A100-class]	inference	312 TFLOP/s	2.04 TB/s	153 FLOPs/B	76
[H100-class]	training	989 TFLOP/s	3.35 TB/s	295 FLOPs/B	148

Compute grew $3\times$*; bandwidth grew* $1.6\times$*. So the faster machine* **demands more arithmetic per byte** *to stay busy.*

Memory Hierarchy in a single Machine

Level	Component	Capacity	Bandwidth	Relative
L1/L2 Cache	On-chip SRAM	50 MB	$\sim$20 TB/s	1$\times$
HBM3	GPU memory	80 GB	3,350 GB/s	6$\times$ slower
System DRAM	DDR5	512 GB–2 TB	300 GB/s	67$\times$ slower
NVMe SSD	Storage	4–30 TB	3.5 GB/s	5,700$\times$ slower

Latency Numbers for ML Systems



Decision rule: If budget $= X$, then $\sum(\text{critical-path stages}) \leq X$.

A latency question means add the time of critical-path stages.

The Iron Law gets Evaluated Twice in Cloud vs Edge

When deciding whether to run a model locally or offload it to the cloud, the Iron Law is evaluated twice: once for local execution, once for remote execution.

$$T_{\text{local}} = \frac{D_{\text{vol}}}{BW_{\text{local}}} + \frac{O}{R_{\text{local}}\eta} + L_{\text{lat}} \qquad T_{\text{remote}} = \underbrace{\frac{D_{\text{vol}}}{BW_{\text{net}}} + L_{\text{net}}}_{T_{\text{net}}} + \frac{O}{R_{\text{remote}}\eta} + L_{\text{lat}}$$

Factory defect detection: 2 MB frame, ResNet-class model.

	Local (Jetson)	**Offload [H100-class]**
Transfer 2 MB @ 1 Gb/s	—	16 ms
Network round trip	—	30 ms
Memory + compute	$\approx$ 0.6 ms	0.015 ms
Total	**0.6 ms**	**46 ms**

Cloud ML: Computational Power

Cloud ML aggregates compute in data centers: **[H100-class]**

Metric	Value
Compute	989 TFLOPS (H100 FP16)
Memory	80 GB HBM3, 3,350 GB/s
Power	700 W per GPU
Latency	100–1000+ ms *end-to-end*
Storage	Petabytes (elastic)

Strengths

Elastic scaling, massive datasets, models too large to run anywhere else

Weaknesses

T_{net} on every request, data leaves the premises, cost

Same H100, two workloads	intensity	balance	
LLM training, large batch	≈ 900	148	**compute-bound**
LLM serving, one token	**1**	148	**bandwidth-starved**

Edge ML: Latency and Privacy

Edge ML moves computation to *local servers*:

Metric	Value
Compute	100 TOPS (Orin NX INT8)
Memory	32 GB LPDDR5, 102 GB/s
Power	15–40 W
Latency	10–100 ms *end-to-end*
Privacy	Data stays on-premises

Iron Law: T_{net} **shrinks**, it does not vanish — WAN round trip becomes a LAN hop.

Use Cases

Factory quality inspection, hospital patient monitoring, retail analytics

Trade-offs

$\approx 10\times$ less compute and $30\times$ less bandwidth than [H100-class]; hardware maintenance

Only **on-device** *inference (mobile, tiny) removes* T_{net} *entirely. An edge server is still a network hop away from the sensor.*

Cloud Vs Edge: Where the Rule Bites?

Cloud training

Not how long *one* batch takes but how many batches per day.

$\rightarrow$ **throughput** $\rightarrow$ **max form**

Raising batch size is the work of *moving* the bottleneck onto the expensive silicon.

Safety control loop (10 ms)

For one task there is one deadline.

$\rightarrow$ **latency** $\rightarrow$ **sum form**

Remember: **Cloud vs. edge is the Iron Law evaluated twice — once with T_{net}, once without.**

Exercise: Autonomous Car — Can the Edge Keep Up?

8 cameras, 30 FPS, 2 MB per frame *[stream → max]*

- 240 inferences/s total
- Orin NX: 102 GB/s peak, $\eta \approx 0.3$
- INT8 model: 25.6 MB weights

Can the bandwidth stage sustain the rate?

Exercise: Autonomous Car — Can the Edge Keep Up?

8 cameras, 30 FPS, 2 MB per frame *[stream → max]*

- 240 inferences/s total
- Orin NX: 102 GB/s peak, $\eta \approx 0.3$
- INT8 model: 25.6 MB weights

Can the bandwidth stage sustain the rate?

Camera data: $8 \times 30 \times 2$ MB $=$ 480 MB/s
Effective BW: $102 \times 0.3 = 30$ GB/s
$\Rightarrow$ **62**$\times$ **headroom.** Ship it?

Exercise: Autonomous Car — Can the Edge Keep Up?

8 cameras, 30 FPS, 2 MB per frame *[stream → max]*

- 240 inferences/s total
- Orin NX: 102 GB/s peak, $\eta \approx 0.3$
- INT8 model: 25.6 MB weights

Can the bandwidth stage sustain the rate?

Camera data: $8 \times 30 \times 2$ MB $=$ 480 MB/s
Effective BW: $102 \times 0.3 = 30$ GB/s
$\Rightarrow$ **62× headroom.** Ship it?

The model.
Activations per inference:
240×25.6 MB $=$ **6.1 GB/s**
$\Rightarrow$ headroom collapses to $\approx 4\times$

Exercise: Autonomous Car — Can the Edge Keep Up?

8 cameras, 30 FPS, 2 MB per frame *[stream → max]*

- 240 inferences/s total
- Orin NX: 102 GB/s peak, $\eta \approx 0.3$
- INT8 model: 25.6 MB weights

Can the bandwidth stage sustain the rate?

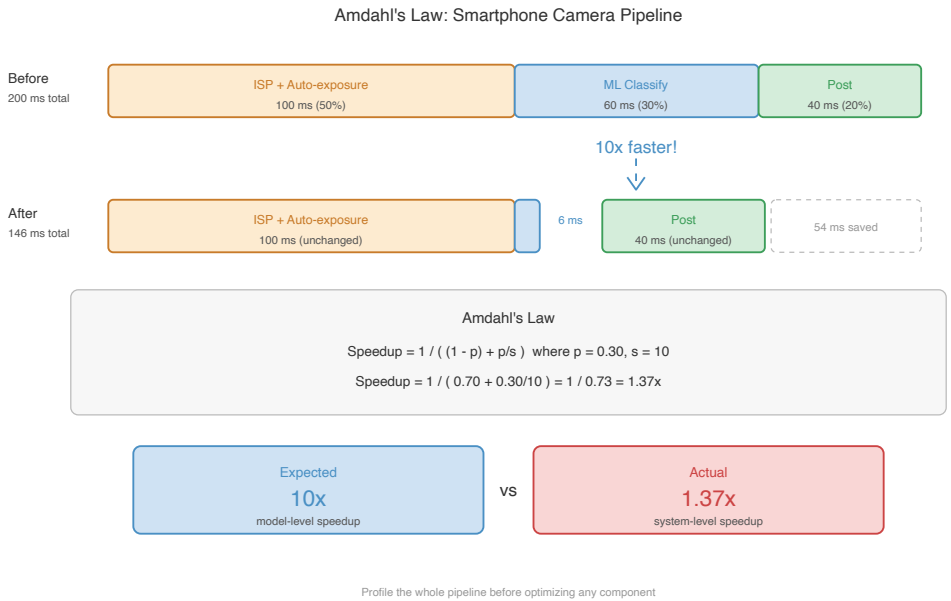
Camera data: $8 \times 30 \times 2$ MB $=$ 480 MB/s
Effective BW: $102 \times 0.3 = 30$ GB/s
$\Rightarrow$ **62× headroom.** Ship it?

The model.
Activations per inference:
240×25.6 MB $=$ **6.1 GB/s**
$\Rightarrow$ headroom collapses to $\approx 4\times$

Batch the 8 cameras:
weights load $30\times$/s $= 0.77$ GB/s
reuse $1 \rightarrow 8$

The model's traffic was $13\times$ *the camera traffic! And "effective bandwidth is 30% of peak" is just a different form of* η*.*

Amdahl's Law: System-Level Speedup



ISP 100 ms + **ML 60 ms** + **post 40 ms.** Now make the model 10× faster:

One photo (sum)	**Continuous capture** (max)
$200 \rightarrow 146$ ms	$\max(100, 60, 40) \rightarrow \max(100, 6, 40)$
1.37× faster	**0% faster**

Amdahl *is* the additive Iron Law:stages that add, where optimizing a non-dominant one buys almost nothing — and in a *pipeline*, nothing at all.

Energy Tax: The Fourth Term

Time is not the only constraint. On mobile and edge devices, **energy** is the hard limit.

Energy Tax:

$$E_{\text{total}} \approx \underbrace{D_{\text{vol}} \times E_{\text{move}}}_{\text{Dominant}} + \underbrace{O \times E_{\text{compute}}}_{\text{Secondary}}$$

Key: Data movement dominates. Minimize D_{vol} to reduce both time *and* energy.

Operation	**Energy**	**Ratio**
FP16 MAC	$\sim$1.1 pJ	1$\times$
INT8 MAC	$\sim$0.2 pJ	0.2$\times$
DRAM byte	$\sim$160 pJ	**145**$\times$

Moving one DRAM byte costs $\approx$**145**$\times$ a FP16 op and $\approx$**800**$\times$ an INT8 op.

Horowitz, "Computing's Energy Problem," ISSCC 2014

The Degradation Equation

$$\text{Accuracy}(t) \approx \text{Accuracy}_0 - \lambda \cdot \mathcal{D}(P_t \| P_0)$$

$\text{Accuracy}(t)$	Accuracy at time t
Accuracy_0	Accuracy at deployment
λ	Model sensitivity to drift
$\mathcal{D}(\cdot\|\cdot)$	Distribution divergence

Key:ML systems degrade through data drift even when code is unchanged.

Exercise: When to Retrain?

When Does the Alarm Fire?

A fraud detector starts at 94% accuracy. It drifts at $\alpha = 0.03$ per month with Δ growing linearly at 0.33 per month. The retraining trigger is 90%. When does it fire?

Exercise: When to Retrain?

When Does the Alarm Fire?
A fraud detector starts at 94% accuracy. It drifts at $\alpha = 0.03$ per month with Δ growing linearly at 0.33 per month. The retraining trigger is 90%. When does it fire?

A fraud detector has:

- $A_0 = 94\%$ accuracy at deployment
- Drift rate: $\alpha \cdot \Delta(t) \approx 1\%$ per month
- Retraining trigger: $A(t) < 90\%$

When does it cross the 90% threshold?

Exercise: When to Retrain?

When Does the Alarm Fire?
A fraud detector starts at 94% accuracy. It drifts at $\alpha = 0.03$ per month with Δ growing linearly at 0.33 per month. The retraining trigger is 90%. When does it fire?

A fraud detector has:

- $A_0 = 94\%$ accuracy at deployment
- Drift rate: $\alpha \cdot \Delta(t) \approx 1\%$ per month
- Retraining trigger: $A(t) < 90\%$

When does it cross the 90% threshold?

Solution:

$A(t) = 0.94 - 0.01t$

Month 1: 93%
Month 2: 92%
Month 3: 91%
Month 4: 90% $\leftarrow$ trigger

Retrain by month 3–4 to stay above SLA.

Return on Compute (RoC)

Return on Compute:

$$\text{cost} \propto \frac{\text{Model} \times \text{Data size}}{\text{H/W Efficiency}}$$

$$\text{RoC} = \frac{\Delta\text{Accuracy}}{\text{Compute Cost}}$$

The economic invariant that governs scaling decisions.

Key insight:The Bitter Lesson says *scale wins*. RoC asks *at what price?*

Illustrative order-of-magnitude estimates:

Approach	**Gain**	**Cost**
GPT-4 pretrain	+5%	$100M
Fine-tune small	+2%	$1K
Distill + quant	+0%	$500

Fine-tuning: 100,000× better RoC than pretraining from scratch.

The Deployment Spectrum

Deployment Spectrum

Paradigm	Where	Latency	Power	Memory	Compute	Best For
Cloud ML	Data centers	100-500	3-5 MW	TB	10^{15} ops/s	Training, complex inference
Edge ML	Local Servers	10-100	100-200 W	GB	10^{14} ops/s	Real-time inference, privacy
Mobile ML	Smartphones	5-50	3-5 W	GB	10^{13} ops/s	Personal-AI, offline
Tiny ML	Microcontrollers	1-10	50-100 mW	KB	10^{9} ops/s	Always-on sensing

Nine orders of magnitude in power (MW to mW) and memory (TB to KB).