End-to-End Machine Learning 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.

Train Vs Deploy

1. How many have trained a model?
2. How many have deployed a model to production?

Train Vs Deploy

1. How many have trained a model?
2. How many have deployed a model to production?

AI is not magic — it operates under a system/infrastructure, and that system/infrastructure has constraints.

The AI Moment

AI is not a future possibility — it is present reality:

- **Voice assistants**: speech → text → response; human conversation gap is ∼200 ms, production systems target 400–600 ms
- **Social media**: billions of ranking requests/day, each completing in ∼60–200 ms
- **Spam Filtering**: classifying billions of emails/day, each in ∼1–5 ms
- **Face ID/fingerprint unlock**: ∼1–2 ms on-device inference with tight power/thermal budget

The AI Moment

AI is not a future possibility — it is present reality:

- **Voice assistants**: speech → text → response; human conversation gap is ∼200 ms, production systems target 400–600 ms
- **Social media**: billions of ranking requests/day, each completing in ∼60–200 ms
- **Spam Filtering**: classifying billions of emails/day, each in ∼1–5 ms
- **Face ID/fingerprint unlock**: ∼1–2 ms on-device inference with tight power/thermal budget

ML + Systems: These real systems have hard requirements and developers must deliver them.

The Gap Between ML Research and Production

A Kaggle notebook is not a production system:

- **60–85%** of ML projects fail to reach production
- 90% of engineering time goes to data + infrastructure
- Model code is ≈**5%** of a production system

Paleyes et al., ACM Computing Surveys 2022

The Gap Between ML Research and Production

A Kaggle notebook is not a production system:

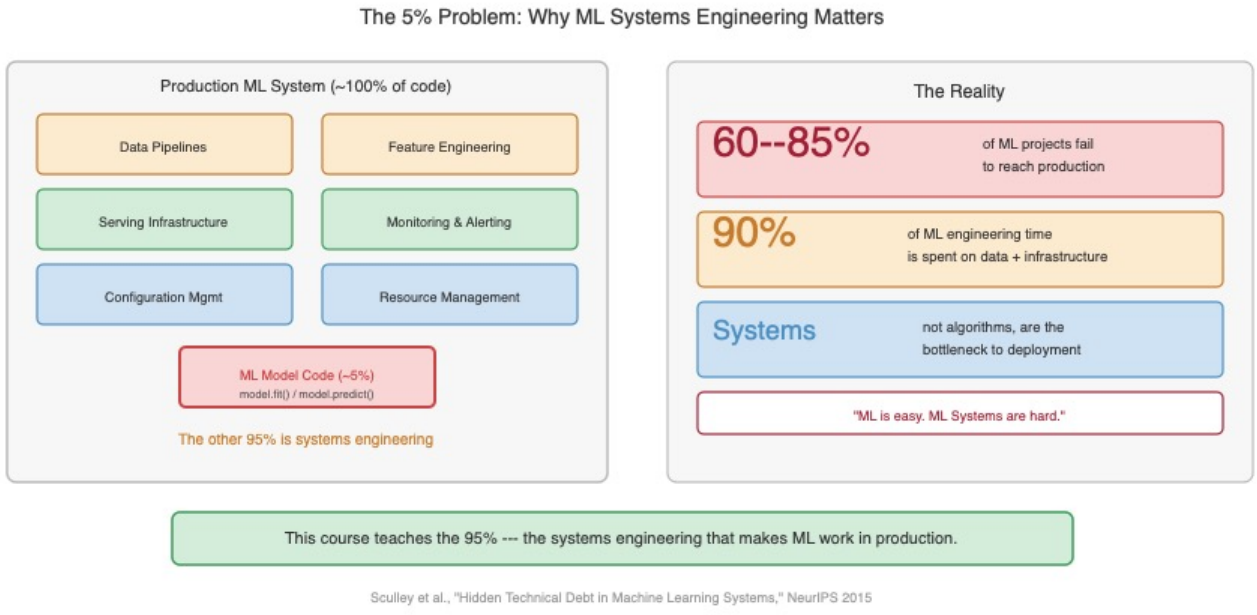
- **60–85%** of ML projects fail to reach production
- 90% of engineering time goes to data + infrastructure
- Model code is ≈**5%** of a production system

	Research	**Production**
Data	Static set	Shifting stream
Hardware	Given GPU	Fleet of tiers
Success	Accuracy	Acc. + SLA
Lifespan	One paper	Years

The bottleneck is systems, not algorithms.

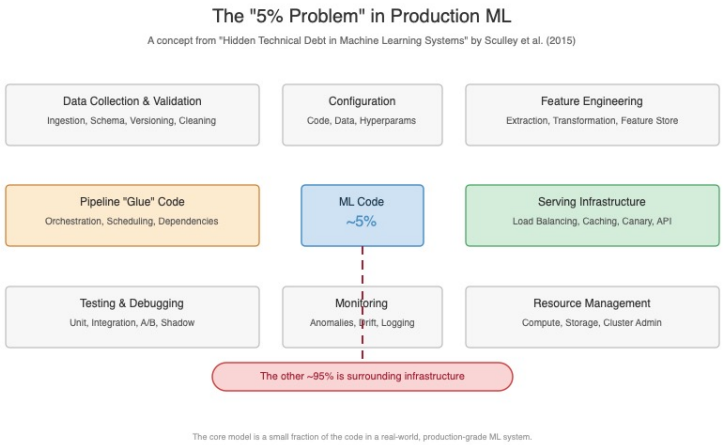
Paleyes et al., ACM Computing Surveys 2022

The 5% Problem



This course teaches the **95%** — the systems engineering that makes ML work.

The 5% Problem



Hidden Technical Debt

In production ML, the model is $\approx$5% of the code. The other 95%:

- Data pipelines & feature engineering
- Serving infrastructure
- Monitoring & configuration

"ML is easy; ML Systems are hard."

Sculley et al., NeurIPS 2015

Why ML Systems Are Different

Why ML Systems Are Different

Traditional software:

- Code bug $\rightarrow$ crash (loud failure)
- Stack trace in milliseconds
- Debugging: trace the code
- Fixed code stays fixed

A function that computed correctly
yesterday computes correctly today.

Why ML Systems Are Different

Traditional software:

- Code bug $\rightarrow$ crash (loud failure)
- Stack trace in milliseconds
- Debugging: trace the code
- Fixed code stays fixed

A function that computed correctly yesterday computes correctly today.

ML systems:

- Data bug $\rightarrow$ wrong prediction (silent failure)
- No exception, no alert
- Debugging: trace the *data*
- Model silently decays as world changes

Sentiment model: 94% on test set $\rightarrow$ 78–82% in production as users adopt new slang. **Code unchanged.**

You cannot test your way to reliability. You must *monitor* it.

Software 1.0 Vs 2.0

Software 1.0

Program: explicit instructions
Bug: trace to a line of code
Update: edit the source file
Test: deterministic paths

Software 1.0 Vs 2.0

Software 1.0

Program: explicit instructions
Bug: trace to a line of code
Update: edit the source file
Test: deterministic paths

Software 2.0

Program: learned weights
Bug: trace to the training data
Update: retrain on new data
Test: monitor production distributions

Systems consequence: **A GPT-4-class model has $\sim$1.8T parameters. The "program" is 3.6 TB of weights. Debugging means auditing the data pipeline, not the Python code.**

Karpathy, 2017

The Bitter Lesson: Scale Beats Expertise

Pattern across 70 years of AI:

- **1950s–70s:** Symbolic AI — logic rules, expert knowledge
- **1980s–90s:** Feature engineering — human-crafted representations
- **2000s–10s:** Statistical ML — learned features, manual tuning
- **2012–now:** Deep learning + massive compute **win**

The Bitter Lesson: Scale Beats Expertise

Pattern across 70 years of AI:

- **1950s–70s:** Symbolic AI — logic rules, expert knowledge
- **1980s–90s:** Feature engineering — human-crafted representations
- **2000s–10s:** Statistical ML — learned features, manual tuning
- **2012–now:** Deep learning + massive compute **win**

"General methods that leverage computation consistently outperform approaches that encode human expertise."
— Sutton, 2019

The Bitter Lesson: Scale Beats Expertise

Pattern across 70 years of AI:

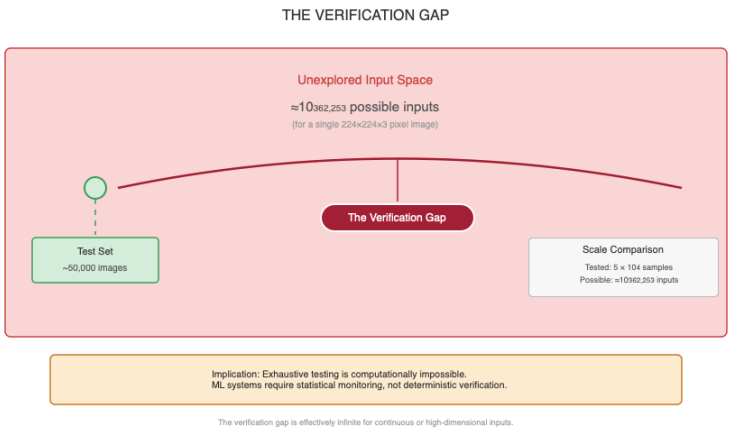
- **1950s–70s:** Symbolic AI — logic rules, expert knowledge
- **1980s–90s:** Feature engineering — human-crafted representations
- **2000s–10s:** Statistical ML — learned features, manual tuning
- **2012–now:** Deep learning + massive compute **win**

"General methods that leverage computation consistently outperform approaches that encode human expertise."
— Sutton, 2019

Systems implication: If scale determines AI progress, then the systems that move compute efficiently—data pipelines, training infrastructure, hardware optimization—are the limiting factor.

This course teaches how to build those systems.

Software 2.0: Missing Piece: Verification



A 224×224 RGB image has:

$256^{150,528}$ possible configurations (a number with **362,000+ digits**)

ImageNet tests only **50,000** of them.

$$\text{Gap} \approx \text{Total Input Space}$$

Consequence:Trade *guaranteed correctness* for *statistical monitoring*.

AI Is Infrastructure

Physical constraints govern ML systems:

- **Memory bandwidth** limits how fast data reaches the processor
- **Power budget** limits where a model can run
- **Speed of light** limits latency over distance
- **Thermodynamics** limits compute per rack

Physics, not elegance, determines what ships.

AI Is Infrastructure

Physical constraints govern ML systems:

- **Memory bandwidth** limits how fast data reaches the processor
- **Power budget** limits where a model can run
- **Speed of light** limits latency over distance
- **Thermodynamics** limits compute per rack

Physics, not elegance, determines what ships.

Example: GPT-4 costs ∼$100M to train — moving 1.8T parameters through memory thousands of times costs that much in electricity and silicon time.

Example: A self-driving car cannot call the cloud — speed of light adds 50+ ms. Physics forces the model onto the edge.

Key: **Every deployment target (phone, car, data center) sits at a different point on these four axes simultaneously, and your architecture has to satisfy all four at once.**

Three Analytical Frameworks

The three analytical frameworks that recur throughout the course:
1. **D-A-M:** Data, Algorithm, Model — where is the bottleneck?
2. **Iron Law:** Latency = Work / Bandwidth — 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 — Volume, quality, distribution
How much data, how fast can we move it?

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

Machine — Hardware throughput, memory
What silicon, what memory hierarchy?

Key:Optimizing one axis shifts pressure to another.

The Data-Algorithms-Machine (DAM) Taxonomy

Every ML system has three interdependent axes:

Data — Volume, quality, distribution
How much data, how fast can we move it?

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

Machine — Hardware throughput, memory
What silicon, what memory hierarchy?

Key:Optimizing one axis shifts pressure to another.

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

Diagnostic: "Which axis is the bottleneck?" answers most ML systems design questions.

The Iron Law of ML Systems

$$T_{\text{total}} = \underbrace{\frac{D_{\text{vol}}}{BW}}_{\text{Data}} + \underbrace{\frac{O}{R_{\text{peak}} \cdot \eta}}_{\text{Compute}} + \underbrace{L_{\text{lat}}}_{\text{Overhead}}$$

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**

The Iron Law of ML Systems

$$T_{\text{total}} = \underbrace{\frac{D_{\text{vol}}}{BW}}_{\text{Data}} + \underbrace{\frac{O}{R_{\text{peak}} \cdot \eta}}_{\text{Compute}} + \underbrace{L_{\text{lat}}}_{\text{Overhead}}$$

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**

Key insight:Every term resolves to **seconds**. The **slowest term dominates**.

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

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.

Example:

Recommendation system: 85% accuracy at launch $\rightarrow$ 79.2% after 6 months. No bugs — just user behavior shift.

Engineering response: Set retraining triggers when $A(t)$ crosses a 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?

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.

Predict: Which Constraint?

A voice assistant must respond in under 200 ms.

What physical constraint dominates?

Memory Bandwidth	Compute Throughput	Network Latency	Power Budget

Five Models

Five models, five bottlenecks

Lighthouse	Bottleneck	Iron Law	Systems Question
ResNet-50	Compute	O/R_p	Is arithmetic fast enough?
GPT-2/Llama	Memory-BW	D/BW	Can memory feed the accelerator?
MobileNetV2	Latency	L_{lat}	Can it meet real-time deadlines?
DLRM	Mem. Capacity	D/BW	Do terabyte-scale embedding tables fit in memory?
KWS	Power	$P \times T$	Can it run on a coin-cell battery?

Same equation, different dominant term. Each model tests a different constraint regime.

Four Parts of Course

Foundations $\rightarrow$ **Development & Training** $\rightarrow$ **Optimization & Acceleration** $\rightarrow$ **Deployment & Operations**

— each part builds on the previous.

3 sprints, 2 of 6 weeks each, 1 of 3 weeks. Each sprint ends with midterm + project presentation, except last where you present the final project.

Course Learning Outcomes

By the end of this course, you will be able to:

1. **Explain** why computational scale outperforms hand-coded expertise (the Bitter Lesson) and why ML systems fail silently
2. **Apply** frameworks to diagnose bottlenecks: data-bound, compute-bound, or machine-bound?
3. **Calculate** memory budgets — can this model fit on this device?
4. **Design** serving pipelines with latency and throughput targets
5. **Evaluate** trade-offs across the deployment spectrum
6. **Compress** models using quantization, pruning, and distillation
7. **Monitor** production systems — detect drift before users do
8. **Make** principled design decisions grounded in physics, not hype

Prerequisites

Required background:

- **Operating systems** — processes, memory, I/O, networking
- **Computer architecture** — caches, pipelines
- **Linear algebra** — matrix multiply, norms
- **Probability** — distributions, Bayes' rule

Helpful but not required:

- PyTorch or TensorFlow experience
- Systems programming (C/C++)
- Cloud computing basics
- Prior ML coursework

Note: We teach ML concepts as needed. No ML expertise required to start.

Course Logistics

Grading (adjust per term)

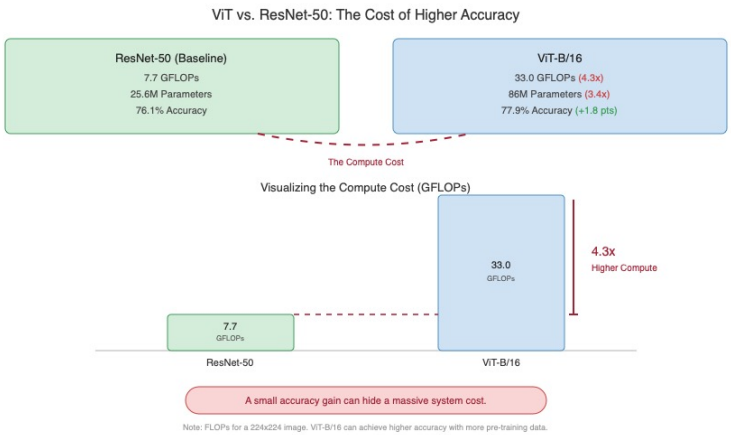
Component	**Weight**
Labs (5-6), Quizzes, classwork	25%
Midterm Exam (2)	30%
Final Project	40%
Participation	5%

Resources

- **Textbook**: *ML Systems*, Vol. I; Designing Machine Learning Systems, Chip Huyen, 2024.
- **Office hours**: Wednesdays 3–4pm, by appointment
- **Discussion forum**: See Canvas

Collaboration policy: Discuss concepts freely. Write your own solutions. Cite all sources including AI.

The Efficiency Counter-Narrative



Two forces co-evolve:

Brute-force scaling

AI compute: 7 orders of magnitude growth

Algorithmic efficiency

44× compute reduction for equivalent accuracy

ViT: 1–2% higher accuracy than ResNet-50, but 4× FLOPs and 3× memory bandwidth. **Better accuracy ≠ better system.**

Return on Compute (RoC)

Return on Compute:

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

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.