



CONFERENCE

—NORTH AMERICA 2026—

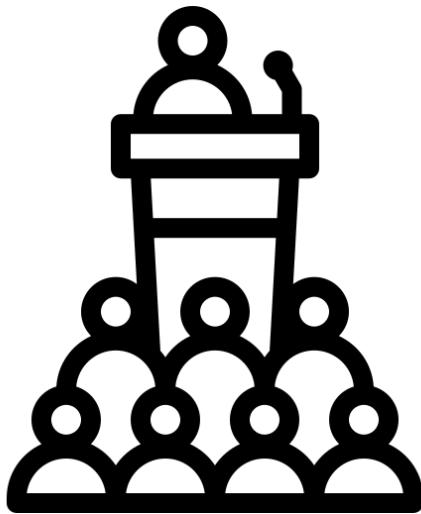
From Scratch to PyTorch: Demystifying ML Frameworks by Building Your Own

**Vijay Janapa Reddi, ETH Zurich & Harvard University
Andrea Mattia Garavagno, ETH Zurich**

Expectations



Prof. Vijay Janapa Reddi, PhD
Harvard University & ETH Zurich
vj@eecs.harvard.edu

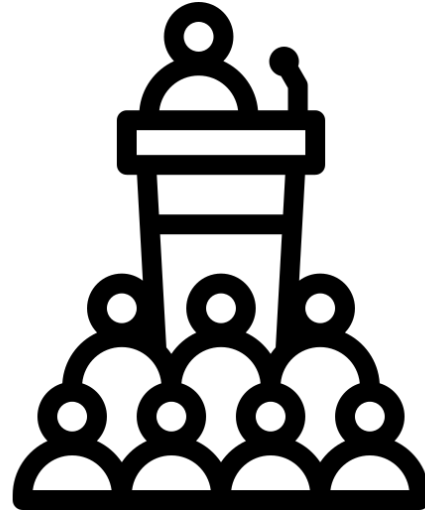


Andrea Mattia Garavagno, PhD
ETH Zurich
agaravagno@ethz.ch

Reality



Andrea Mattia Garavagno, PhD
ETH Zurich
agaravagno@ethz.ch



TinyTorch



Don't import it. Build it.

Build your own ML framework: from tensors to systems.

Preview · Classroom release Spring 2027

An educational framework for building and optimizing ML: understand how PyTorch, TensorFlow, and JAX really work.

TinyTorch is usable today for self-paced learning and active course pilots. APIs, instructor packaging, and classroom workflows will continue to stabilize through the Spring 2027 classroom release. Self-study works today; courses adopting TinyTorch before then should expect to pilot it.

Start Building →

☆ 29,051 stars on GitHub: **add yours** and support free ML education

Why Build Instead of Use?



"Building systems creates irreversible understanding."

Traditional ML Education

```
import torch
model = torch.nn.Linear(784, 10)
output = model(input)
# When this breaks, you're stuck
```

Problem: You can't debug what you don't understand.

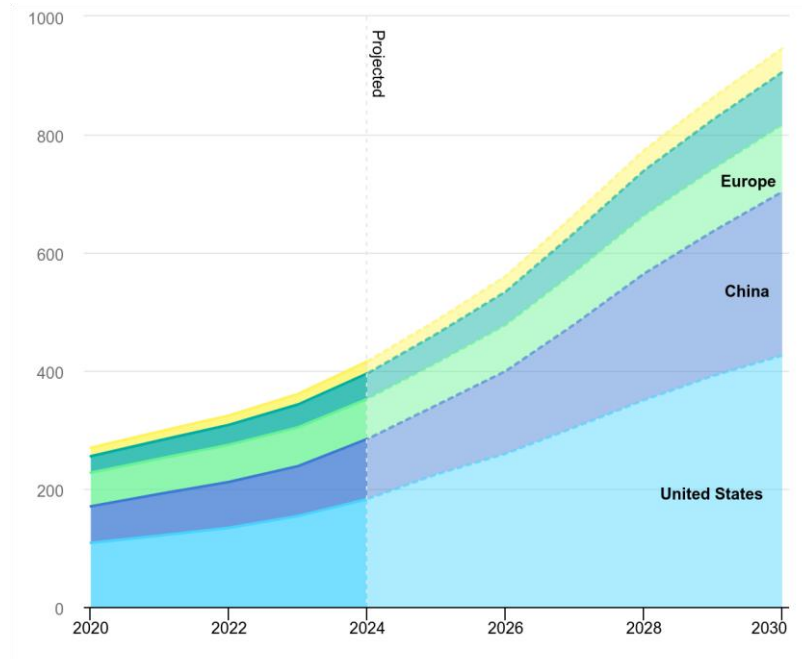
TinyTorch: Build → Use → Reflect

```
# BUILD it yourself
class Linear:
    def forward(self, x):
        return x @ self.weight + self.bias

# USE it on real data
loss.backward() # YOUR autograd
```

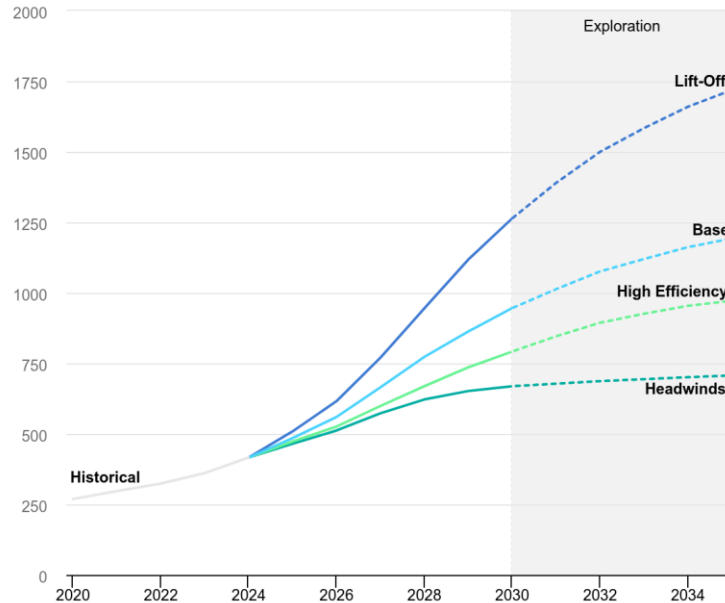
Advantage: You can debug it because you built it.

A Critically Needed Understanding



Data center electricity consumption by region, Base Case, 2020-2030

A Critically Needed Understanding



Global data center electricity consumption by sensitivity case, 2020-2035

Recreate ML History Through Milestones



1958

The Perceptron

The first neural network, run
forward with your layers

Input → Linear → Sigmoid →
Output

Learning Path



Four progressive tiers take you from foundations to production systems:

Foundation (01-08)

Tensors, autograd, layers, training loops

Architecture (09-13)

CNNs, attention, transformers, GPT

Optimization (14-19)

Profiling, quantization, acceleration

Torch Olympics (20)

Competition-ready capstone project

With PyTorch's Syntax



TinyTorch

```
#tinytorch  
for batch_images, batch_labels in train_loader:  
    # Compute prediction and loss  
    logits = model(batch_images)  
    loss = loss_fn(logits, batch_labels)  
    # Backpropagation  
    loss.backward()  
    optimizer.step()  
    optimizer.zero_grad()
```

PyTorch

```
#pytorch  
for batch, (X, y) in enumerate(dataloader):  
    # Compute prediction and loss  
    pred = model(X)  
    loss = loss_fn(pred, y)  
    # Backpropagation  
    loss.backward()  
    optimizer.step()  
    optimizer.zero_grad()
```

Is This For You?



Students

Taking ML courses, want to understand what's behind

```
import torch
```

Instructors

Teaching ML systems with ready-made hands-on labs

Self-learners

Career changers or hobbyists going deeper than tutorials

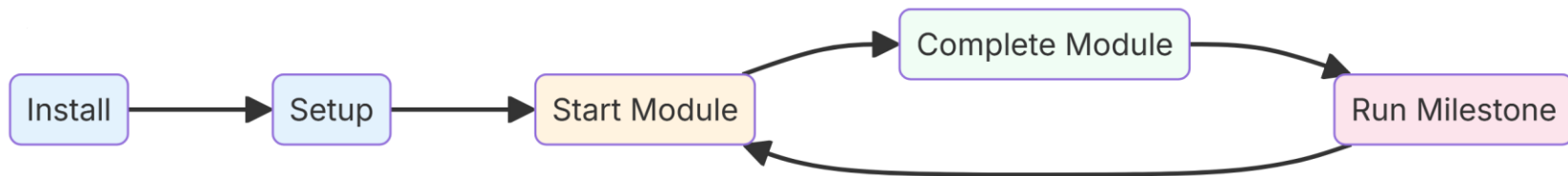
Prerequisites: Python + basic linear algebra. No ML experience required.



Let's Dive Into TinyTorch

Attention: CLI Ahead

Workflow



Easy Installation & Setup



macOS / Linux

Windows

Development Preview (dev branch)

```
curl -sSL mlsysbook.ai/tinytorch/install.sh | bash
cd tinytorch
source .venv/bin/activate
tito setup
tito system health
```



🔥 Tiny🔥Torch Setup Complete!

🎉 Tiny🔥Torch setup completed successfully!

👋 Welcome, andre!

✉ Email: dev@tinytorch.local

🏠 Affiliation: Independent

💻 Platform: Linux

🐍 Python: 3.12

🔥 Activate your environment:

```
source .venv/bin/activate # On Windows: .venv\Scripts\activate
```

🚀 Start building ML systems:

```
tito module start 01 # Begin with tensor foundations
```

💡 Essential commands:

- `tito system health` – Check environment
- `tito module status` – Track progress

```
(.venv) andre@hp-andrea:~/tinytorch$ tito module start 01|
```

Notebook – Introduction



Module 01: Tensor Foundation - Building Blocks of ML

Welcome to Module 01! You're about to build the foundational Tensor class that powers all machine learning operations.

Prerequisites & Progress

You've Built: Nothing - this is our foundation! **You'll Build:** A complete Tensor class with arithmetic, matrix operations, and shape manipulation **You'll Enable:** Foundation for activations, layers, and all future neural network components

Connection Map:

NumPy Arrays → Tensor → Activations (Module 02)
(raw data) (ML ops) (intelligence)

Learning Objectives

By the end of this module, you will:

1. Implement a complete Tensor class with fundamental operations
2. Understand tensors as the universal data structure in ML
3. Master broadcasting, matrix multiplication, and shape manipulation
4. Test tensor operations with immediate validation

Let's get started!

Notebook – The Code Skeleton



```
#!/ export
class Tensor:
    """Educational tensor - the foundation of machine learning computation.

    This class provides the core data structure for all ML operations:
    - data: The actual numerical values (NumPy array)
    - shape: Dimensions of the tensor
    - size: Total number of elements
    - dtype: Data type (float32)

    All arithmetic, matrix, and shape operations are built on this foundation.
    """

    def __init__(self, data):
        """Create a new tensor from data.
```

Notebook – Methods Already Done



```
def __add__(self, other):  
    if isinstance(other, Tensor):  
        return Tensor(self.data + other.data)  
    else:  
        return Tensor(self.data + other)
```

Notebook – Methods To Be Done



```
def matmul(self, other):  
    """Matrix multiplication of two tensors.  
  
    Validates shapes via _validate_matmul_shapes, then computes the product.  
    For 2D matrices, uses explicit nested loops so you can see exactly how  
    each output element is a dot product of a row and a column. For batched  
    (3D+) inputs, delegates to np.matmul.  
  
    TODO: Validate inputs with _validate_matmul_shapes, then compute the  
    matrix product using explicit loops for 2D and np.matmul for 3D+.  
  
    APPROACH:  
    1. Call self._validate_matmul_shapes(other) to check compatibility  
    2. For 2D matrices: use explicit nested loops with np.dot per element  
    3. For batched (3D+): use np.matmul for correctness  
    4. Return result wrapped in Tensor
```

Notebook – Unit Testing For Self-Learning



```
def test_unit_matrix_multiplication():
    """🧪 Test matrix multiplication operations."""
    print("🧪 Unit Test: Matrix Multiplication...")

    # Test 2x2 matrix multiplication (basic case)
    a = Tensor([[1, 2], [3, 4]]) # 2x2
    b = Tensor([[5, 6], [7, 8]]) # 2x2
    result = a.matmul(b)
    # Expected: [[1*5+2*7, 1*6+2*8], [3*5+4*7, 3*6+4*8]] = [[19, 22], [43, 50]]
    expected = np.array([[19, 22], [43, 50]], dtype=np.float32)
    assert np.array_equal(result.data, expected)
```

```
(.venv) andre@hp-andrea:~/tinytorch$ tito module complete 01|
```

Community Ecosystem





The First Milestone

After Some Modules...

```
(.venv) andre@hp-andrea:~/tinytorch$ tito module status
```

```
(.venv) andre@hp-andrea:~/tinytorch$ tito milestone run 01|
```

Your Journey



Four progressive tiers take you from foundations to production systems:

Foundation (01-08)

Tensors, autograd, layers, training loops

Architecture (09-13)

CNNs, attention, transformers, GPT

Optimization (14-19)

Profiling, quantization, acceleration

Torch Olympics (20)

Competition-ready capstone project

GPT's Notebook – Method To Be Done



```
def forward(self, tokens):  
    """  
    Forward pass through GPT model.  
  
    TODO: Implement the complete GPT forward pass  
  
    APPROACH:  
    1. Get token embeddings and positional embeddings  
    2. Add them together (broadcasting handles different shapes)  
    3. Pass through all transformer blocks sequentially  
    4. Apply final layer norm and language modeling head  
  
    COMPUTATION FLOW:  
    tokens → embed + pos_embed → blocks → ln_f → lm_head → logits  
  
    CAUSAL MASKING:  
    For autoregressive generation, we need to prevent tokens from  
    seeing future tokens. This is handled by the attention mask.  
  
    HINT: Create position indices as range(seq_len) for positional embedding  
    """
```

GPT's Milestone

Sequence Reversal

CHALLENGE 1: SEQUENCE REVERSAL

Examples:

ZPTBWV -> VWBTPZ

BLSSLB -> BLSSLB

ECWKEO -> OEKWCE

Press Enter to continue...

DataLoader: 600 samples, batch_size=16, 38 batches

Training...  100% 0:00:03

PASSED! Accuracy: 100.0% (target: 95%)

Press Enter to continue...

Sample predictions:

OZNOTK -> KTONZO

GWGPFE -> EFPGWG

IQTN YM -> MYNTQI

SZWZUO -> OUZWZS

KLRNTD -> DTNRLK

MLPerf's Notebook – Method To Be Done



```
def update(self, layer_idx: int, key: Tensor, value: Tensor) -> None:
    """
    Update cache with new key-value pairs for given layer.

    TODO: Efficiently append new K,V to cache without data copying

    APPROACH:
    1. Validate layer_idx is in range [0, num_layers-1]
    2. Validate seq_pos hasn't exceeded max_seq_len
    3. Retrieve the (key_cache, value_cache) tuple for this layer
    4. Write new key to position seq_pos in key_cache using indexed assignment
    5. Write new value to position seq_pos in value_cache using indexed assignment
    6. Note: seq_pos is advanced externally via advance() after all layers

    This is the core caching operation - efficiently append new K,V
    to the cache without recomputation. This operation is O(1) because
    it's just an indexed assignment.

    IMPORTANT: KV caching is designed for INFERENCE (generation) only,
    not training. During generation, gradients are not computed. If you
    need gradients, don't use caching (use standard forward pass instead).

    Args:
        layer_idx: Which transformer layer (0 to num_layers-1)
        key: New key tensor, shape (batch_size, num_heads, 1, head_dim)
        value: New value tensor, shape (batch_size, num_heads, 1, head_dim)
```

MLPerf's Milestone

KV Cache Perf

⚡ GENERATION SPEEDUP RESULTS

🎯 KV Cache Performance

Mode	Total Time	Per Token	Speedup
🐌 Without Cache	3.1 ms	0.19 ms	1×
⚡ With YOUR KVCache	0.3 ms	0.02 ms	11.0×

Press Enter to continue...

💾 THE TRADEOFF: Speed vs Memory

Cache Memory Used: 8.00 KB

Why is this worth it?

- Generation is 11.0× faster
- Memory cost is small (8.0 KB)
- For GPT-2 (1.5B params), cache is ~1% of model size
- **Speed gain >> Memory cost**

This is why ALL production LLMs use KV caching!

Still, Something Is Missing



1998

CNN Revolution

A LeNet-style convolutional network on handwritten digits

Images → Conv → Pool → ... →
Classes

2017

Transformer Era

Train TinyGPT on Shakespeare,
TinyCopilot code generation, and
concept Q&A

Tokens → Embeddings → Causal
Attention + MLP → Sampling

Help Us Fill the Gaps: Extend TinyTorch



Key Takeaways

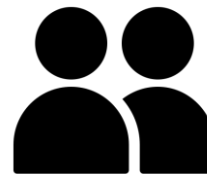


- An **educational project**
- For **understanding** what's behind **ML frameworks**
- By using **PyTorch's syntax**
- To **design better ML systems**

Call To Action



Use ML Framework



Build ML Framework

Call To Action



Use ML Framework



Build ML Framework

Call For Action



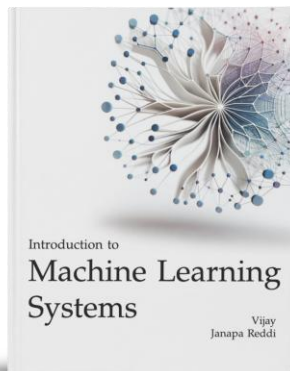
Try it! 
We want your
Feedback!



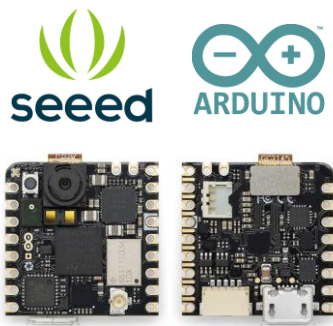
More Than TinyTorch



MLSysbook.ai



Text Books



**Hardware
Kits**

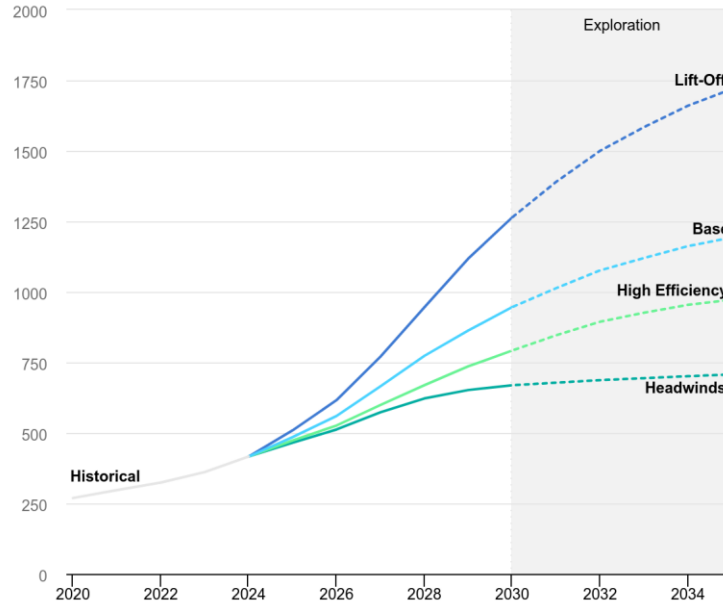


**ML
Framework**

Not Just Energy Efficiency



MLSysbook.ai

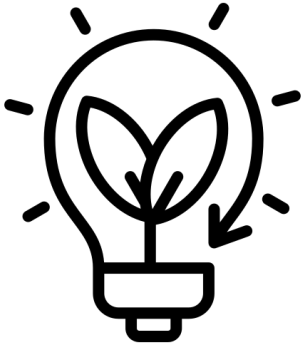


Global data center electricity consumption by sensitivity case, 2020-2035

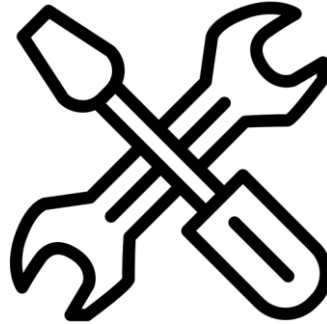
More Than Energy Efficiency



MLSysbook.ai



Efficiency



Reliability

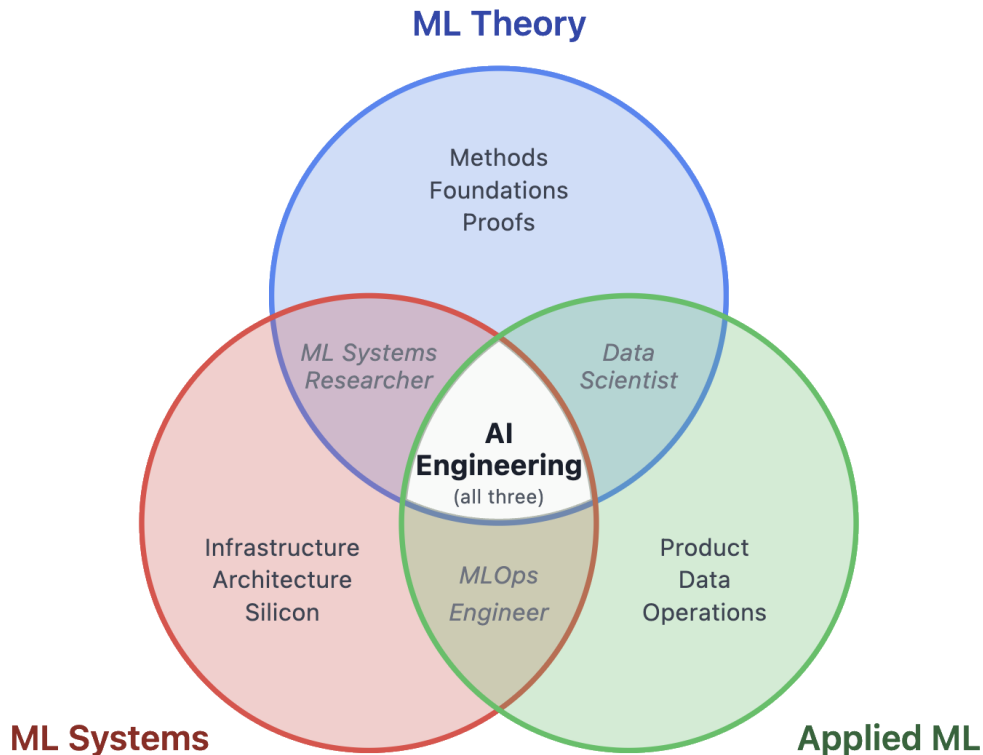


Scale

Promoting AI Engineering



MLSysbook.ai



Which Is Nothing New

SOFTWARE ENGINEERING



MLSysbook.ai

Report on a conference sponsored by the
NATO SCIENCE COMMITTEE
Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer
Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

Editors: Peter Naur and Brian Randell

A Complete Curriculum



MLSysbook.ai

The image shows the cover of the 'Machine Learning Systems' textbook and a preview of its two volumes. The main cover on the left is titled 'Machine Learning Systems' and describes it as 'The physics of AI engineering.' It mentions a rigorous, principles-first treatment of ML systems. Below the title, it states 'Harvard University · MIT Press 2026' and 'Actively maintained · Last updated October 2026'. To the right, two volume covers are shown: 'VOLUME I: Introduction to Machine Learning Systems' and 'VOLUME II: Machine Learning Systems at Scale'. Both volumes are by Vijay Janapa Reddi and offer HTML, PDF, and EPUB formats. The entire graphic is set against a background of colorful, scattered squares.

TWO-VOLUME TEXTBOOK

Machine Learning Systems.

The physics of AI engineering.

A rigorous, principles-first treatment of how ML systems are built, optimized, and deployed — from a single machine to fleet-scale infrastructure.

Harvard University · MIT Press 2026

● Actively maintained · Last updated October 2026 · [Release notes](#)

VOLUME I
Introduction to Machine Learning Systems
HTML PDF EPUB

VOLUME II
Machine Learning Systems at Scale
HTML PDF EPUB

EXPLORER THE CURRICULUM

A Complete Curriculum



MLSysbook.ai

FOR STUDENTS & LEARNERS

Lab 17 - The Fleet Synthesis

Explore



EXPLORE

Labs

Interactive Marimo notebooks. Change a parameter, see what breaks, build intuition.

tinytorch - tensor.py

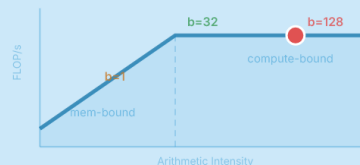
```
class Tensor:
    def __init__(self, data):
        self.data = data
        self.grad = 0.0
        self._backward = lambda: None
```

BUILD

TinyTorch

Build your own ML framework from scratch across 20 progressive modules. Zero magic.

\$ mlsysim eval Llama3_70B H100 --batch-size 1

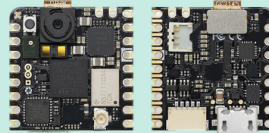


MODEL

MLSys·im

First-principles performance modeling. One command, every bottleneck.

Arduino · Seed · Grove · Raspberry Pi



DEPLOY

Hardware Kits

Deploy ML to Arduino, Seed, Grove, and Raspberry Pi. Real memory limits, real power budgets.

A Complete Curriculum



MLSysbook.ai

FOR CAREER & INSTRUCTORS

Systems Design L5 · Staff

A 70B model needs 1,000 req/s.
Walk through your hardware selection and parallelism strategy.

Hardware

Parallelism

Trade-offs

Cloud Edge Mobile TinyML

PRACTICE

StaffML

Physics-grounded interview questions for ML systems roles.
Vault, drills, and mock interviews.

The Blueprint — Course Architecture
ML Systems · Two-Semester Curriculum

Semester 1: Foundations 16 wks · Vol I · 8 assignments	Semester 2: At Scale 16 wks · Vol II · capstone
Assessment Rubrics · Peer review · Grading	Teaching Staff Pedagogy · TA handbook

READY

ADOPT

Instructor Hub

The AI Engineering Blueprint: two-semester syllabi, pedagogy guide, rubrics, and TA handbook.

Intro Systems DNN **Training** Accel Deploy Ethics

The Iron Law of ML Systems

$$T = D/BW + O/(R \cdot \eta) + L$$

- **Data Term** — memory bandwidth
- **Compute Term** — utilization $\eta \leq 0.7$
- **Latency Term** — orchestration overhead

Harvard University · ML Systems 12 / 38

TEACH

Lecture Slides

35 Beamer decks with speaker notes and 280 original SVG diagrams. Drop in and teach.

MLSysBook Weekly 4

- New: Vol II Ch. 7 — Fault Tolerance
- Updated: TinyTorch Module 20
- Community: 1,200+ PRs merged
- Milestone: 28,000+ GitHub stars

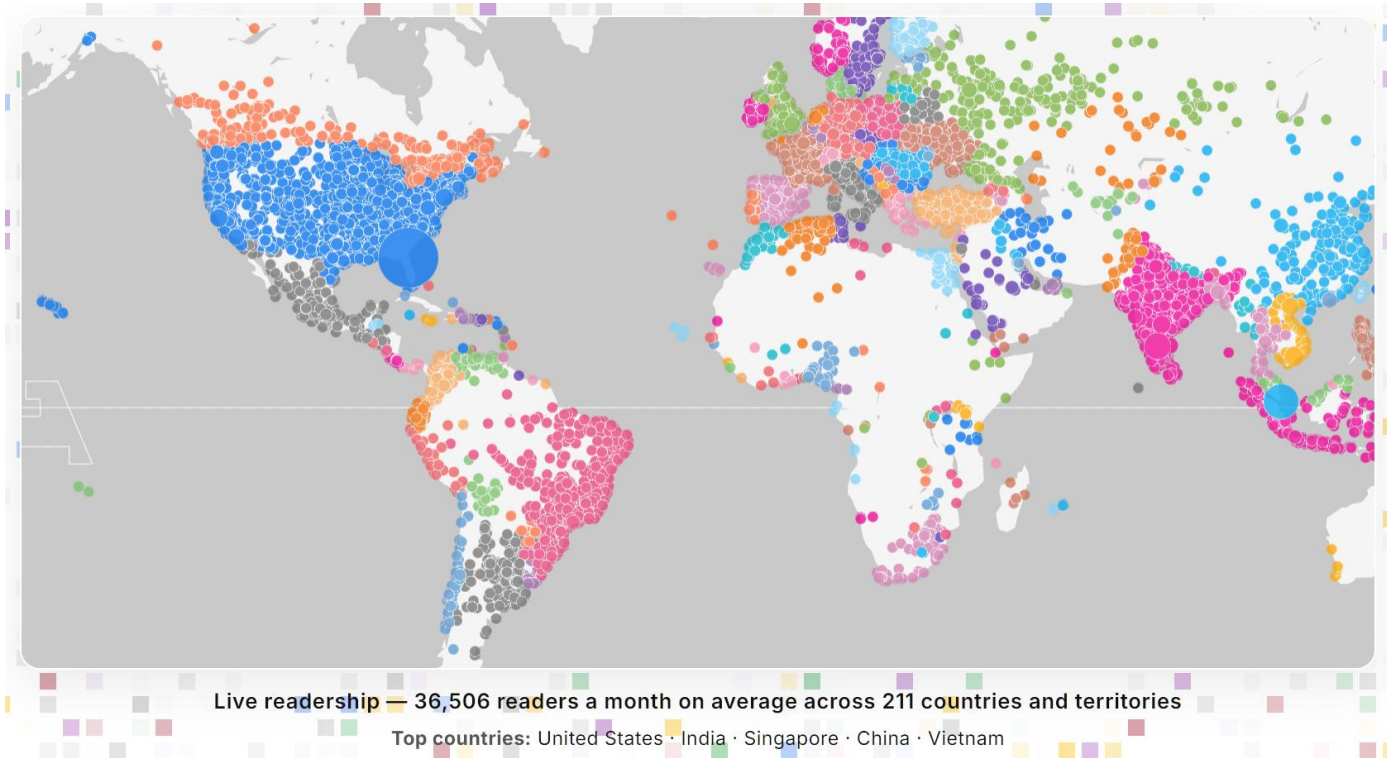
Join 1,100+ subscribers

FOLLOW

Newsletter

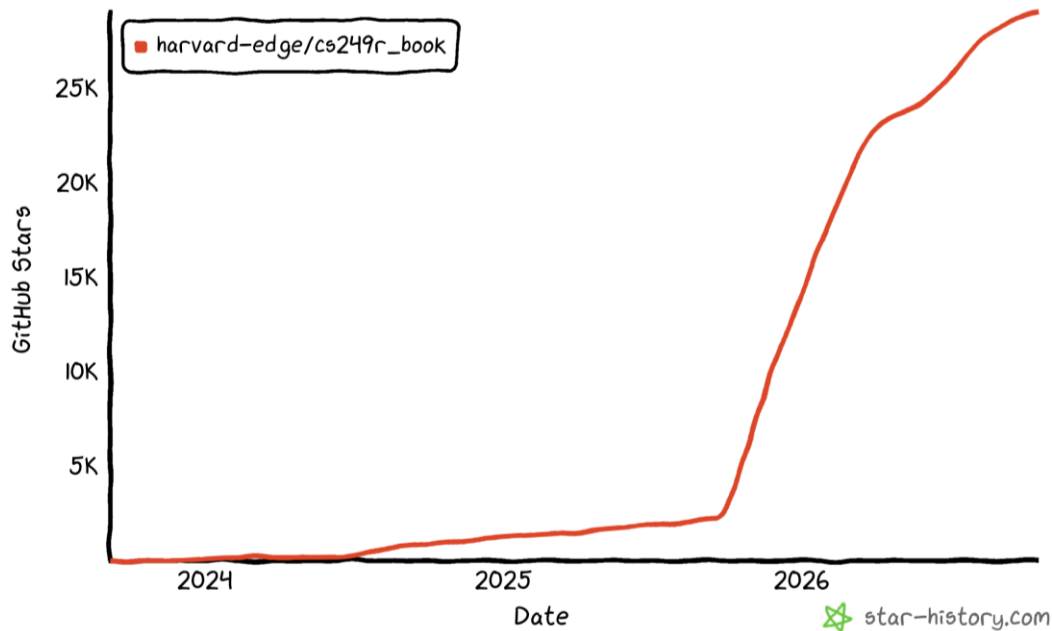
Updates on the curriculum, new chapters, and what the community is building.

A Community Without Frontiers





Star History



With a Clear Goal

OUR MISSION

**AI education should be
free and open to everyone.**

Everyone calls AI the new electricity — but electricity is useless without engineers who can build the grid. For AI to be efficient, reliable, and safe, the world needs engineers who understand how to build it.

That knowledge should be accessible to anyone willing to learn. This curriculum is our commitment to making it so.

Call For Action



MLSysbook.ai



MLSysbook.ai



Prof. Vijay Janapa Reddi, PhD
Harvard University & ETH Zurich
vj@eecs.harvard.edu



Andrea Mattia Garavagno, PhD
ETH Zurich
agaravagno@ethz.ch