MLSys·im Tutorial — Module 1

Foundations & Architecture

Vijay Janapa Reddi

Conference Tutorial

Harvard University

Roadmap: Conference Tutorial

Module	Topic	Status
Module 1	**Foundations & Architecture**	**← You are here**
Module 2	Advanced Single-Node Analysis	
Module 3	Scale, Dollars, and Carbon	
Module 4	Design Space Exploration & Synthesis	

TUTORIAL | MODULE 1

The Problem: Why Analytical Modeling?

1

The ML Systems Complexity Explosion

- **Scale:** Models are growing 10x per year. Clusters span 100,000+ GPUs.
- **Heterogeneity:** GPUs, TPUs, LPU, custom ASIC architectures.
- **Metrics:** Performance is no longer just "latency". It's Throughput, TTFT, ITL, TCO ($ / token), and Carbon Footprint (tonnes CO_2e).

The Cycle-Accurate Simulator Trap

Detailed cycle-level simulators (e.g., gem5) are too slow to sweep 10,000 distributed cluster configurations. We need tools that provide **first-principles, analytical insights** instantly.

TUTORIAL | MODULE 1

Architecture & Registries

The `mlsysim` Philosophy: Demand vs. Supply

Strict Separation of Concerns:

- **Demand (Workloads):** How many FLOPs? How many bytes of weights? (Abstract mathematical operations).
- **Supply (Hardware):** What is the peak TFLOP/s? What is the HBM bandwidth? (Physical silicon constraints).

Solvers sit in the middle, evaluating the intersection to find the *binding constraint*.

The Type System & Registries

`mlsysim` uses a strict, typed registry system powered by Pydantic. No magic dictionaries.

```python
from mlsysim.hardware.types import HardwareNode, ComputeCore, MemoryHierarchy
from mlsysim.core.units import TFLOPs, GB, TB, second, watt

# Defining an accelerator using strict physical quantities
speculative_gpu = HardwareNode(
    name="Speculative GPU",
    release_year=2027,
    compute=ComputeCore(peak_flops=1200 * TFLOPs / second),
    memory=MemoryHierarchy(capacity=144 * GB, bandwidth=5.0 * TB / second),
    tdp=800 * watt
)
```

Zero Hallucinations: The Provenance Audit

Academic tools must be reproducible. Every vetted number in `mlsysim` is bound to a **Provenance Record**.

- **Provenance is metadata:** It records how we know a number, not where the number lives.
- **Semantic homes:** Hardware specs live in `Hardware`; nodes/racks/fleets in `Systems`; grids/prices in `Infrastructure`.
- **Narrative anchors:** Cited scalar results live in `Literature`; reusable teaching scenarios live in `Scenarios`.

`audit_provenance` verifies every registry value has traceable lineage.

Dimensional Strictness (`pint`)

The Problem:

```
# Is this MB/s or GB/s?
# Wait, bits or bytes?
bandwidth = 400
```

The `mlsysim` Solution:

```
# Will throw runtime error if
# divided by seconds instead of bits
bw = Q_("400 Gbps")
speed = bw.to("GB/s") # 50 GB/s
```

Every API boundary strictly enforces SI units at runtime. This prevents silent mismatches when mixing networking (bits) and memory (bytes).

TUTORIAL | MODULE 1

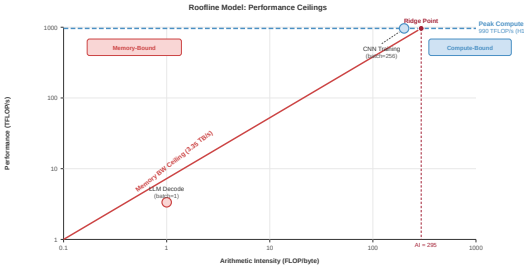
Iron Law & Roofline (Tier 1)

The ML Iron Law

$$\text{Time} = \frac{\text{Operations}}{\text{Peak Performance} \times \text{Utilization}}$$

- **Operations:** From the Model Registry (e.g., Llama 3 8B).
- **Peak Performance:** From the Hardware Registry (e.g., H100 dense TFLOP/s).
- **Utilization (MFU):** The fraction of peak actually achieved.

The Roofline Model



Arithmetic Intensity determines if you are **Memory Bound** (sloped roof) or **Compute Bound** (flat roof).

Live Demo: Tier 1 Execution

```python
from mlsysim.engine.engine import Engine
from mlsysim.hardware.registry import Hardware
from mlsysim.models.registry import Models

model = Models.Language.Llama3_8B
hardware = Hardware.Cloud.H100

# Run the Tier 1 SingleNodeModel solver
perf = Engine.solve(model, hardware, batch_size=1, precision="fp16")

print(f"Latency: {perf.latency.to('ms'):.1f}")
print(f"Bottleneck: {perf.bottleneck}")
```

Summary: Module 1

1. `mlsysim` provides **first-principles, analytical reasoning**.
2. **Strict Separation:** Demand (Workloads) vs Supply (Hardware).
3. **Reproducibility:** Driven by Registries, Provenance, and Dimensional Strictness (`pint`).
4. **Tier 1 Execution:** Powered by the Iron Law and Roofline Model.

Next up: Module 2 - Advanced Single-Node Analysis!