# MLSys·im Tutorial — Module 2

## Advanced Single-Node Analysis

Vijay Janapa Reddi

Conference Tutorial

Harvard University

# Roadmap: Conference Tutorial

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

TUTORIAL | MODULE 2

# The Data Wall

1

# Beyond FLOPs: The Data Wall

GPUs are so fast they often starve waiting for data.

- **Ingestion (I/O):** Can the storage sub-system (NVMe, PCIe) push bytes fast enough?
- **Transformation (CPU):** Can the CPUs decode JPEGs and run augmentations fast enough to keep the GPUs busy?

# Live Demo: Uncovering the Data Wall

```python
from mlsysim.solvers import DataModel, TransformationModel
from mlsysim.hardware.registry import Hardware
from mlsysim.core.units import Q_

demand_rate = Q_("40000 1/s") * Q_("150 KB") # ~6 GB/s

# 1. Check if the DGX A100 PCIe bus can handle the bandwidth
data_result = DataModel().solve(
    workload_data_rate=demand_rate, hardware=Hardware.Cloud.A100)
print(f"I/O Stalled: {data_result.is_stalled}") # False

# 2. Check if the CPUs can decode/augment images fast enough
transform_result = TransformationModel().solve(
    batch_size=40000,
    cpu_throughput_per_worker_hz=850, # JPEG decode + crop
    num_workers=64                    # 8 CPUs per GPU
)
print(f"CPU Stalled: {transform_result.is_bottleneck}") # True
```

TUTORIAL | MODULE 2

# LLM Serving & Memory Walls

2

# The Two Phases of LLM Serving

LLM inference is not a single mathematical operation; it is a stateful process with two distinct physical regimes.

**1. Pre-fill (Prompt Processing)**

- Processes all prompt tokens in parallel.
- Matrix-Matrix multiplication.
- High arithmetic intensity $\Rightarrow$ **Compute Bound**.
- Metric: Time-to-First-Token (TTFT).

**2. Decoding (Token Generation)**

- Generates one token at a time autoregressively.
- Matrix-Vector multiplication.
- Low arithmetic intensity $\Rightarrow$ **Memory Bound**.
- Metric: Inter-Token Latency (ITL).

TUTORIAL | MODULE 2

# Algorithmic Optimizations

3

# Speculative Decoding

*Use a smaller draft model to propose tokens, then verify with the target model.*

```python
from mlsysim.solvers import ServingModel
from mlsysim.hardware.registry import Hardware
from mlsysim.models.registry import Models

target = Models.Language.Llama3_70B
draft = Models.Language.Llama3_8B
hardware = Hardware.Cloud.H100

solver = ServingModel()
base = solver.solve(target, hardware, seq_len=2048, batch_size=1)

spec = solver.solve(
    target, hardware, seq_len=2048, batch_size=1,
    draft_model=draft, draft_acceptance_rate=0.75)
print(f"Speedup: {base.itl / spec.itl:.2f}x")
```

TUTORIAL | MODULE 2

# The Reasoning Wall

4

# Wall 12: Inference-Time Compute

With models like OpenAI o1, compute scaling shifts from training to inference. The model generates $K$ hidden reasoning steps before answering.

```python
from mlsysim.solvers import InferenceScalingModel

reasoning_solver = InferenceScalingModel()
result = reasoning_solver.solve(
    model=Models.Language.Llama3_8B, hardware=Hardware.Cloud.H100,
    reasoning_steps=128,  # K hidden CoT steps
    precision="fp16", efficiency=0.5)

print(f"Reasoning Time: {result.total_reasoning_time.to('s'):.1f}")
print(f"Energy per Query: {result.energy_per_query.to('J'):.1f}")
```

**Impact:** Inference shifts back toward being *compute-bound* as generation length dominates prefill.

# Summary: Module 2

1. **Data Wall:** Real-world throughput is often bottlenecked by CPU transformation, not GPU FLOPs.
2. **Serving:** Pre-fill is Compute-bound; Decoding is Memory-bound.
3. **Algorithmic Optimizations:** We can instantly model the speedup of Speculative Decoding and Quantization.
4. **Reasoning Wall:** Hidden CoT tokens push inference back into the compute-bound regime.

*Next up: Module 3 - Scale, Dollars, and Carbon!*