Performance Benchmarking of Neural Networks Across CPU and GPU Environments

Ruturaj Tambe

Abstract

This report evaluates the performance of neural network workloads across local CPU, cloud CPU, and GPU environments. Using four neural network architectures and a standardized training configuration, we measure runtime, throughput, achieved FLOPs, and efficiency. Roofline models are constructed to analyze compute vs. memory bottlenecks, and a batch-size sweep experiment is conducted to study scaling effects.

Contents

1 Experiment Design

1.1 Objective

The objective is to compare the performance of neural network training workloads across different compute environments (local CPU, cloud CPU, GPU). The experiment measures throughput, latency, memory usage, and achieved FLOPs for multiple neural network models.

1.2 Hypothesis

- GPUs will deliver significantly higher throughput than CPUs, especially for deeper models with high arithmetic intensity.
- CPU vs GPU speedup will vary depending on model complexity.
- Increasing batch size should improve GPU utilization and move the workload closer to the compute roofline.

1.3 Experimental Scenarios

- **Scenario A: Environment Comparison** Four models run under identical configurations on:
  - Local CPU
  - Cloud CPU
  - GPU (local/cloud)
- **Scenario B: Batch Size Sweep** Model 4 is run with multiple batch sizes on local machine to analyze scaling.

1.4 Experimental Controls

To ensure that the comparisons across environments were scientifically valid, the following controls were enforced:

- **Identical dataset**: Tiny-ImageNet was used across all runs. It is large enough to stress compute and memory, yet small enough for repeated benchmarking.
- **Same training hyperparameters**: 1 epoch, same optimizer and learning rate, and identical transforms on every device to avoid confounders.
- **Consistent batch sizes**: The same batch sizes were used across environments, except where instability occurred (e.g., SqueezeNet at large batch sizes), in which case the smallest stable batch was used consistently.
- **Same code path and PyTorch family**: The same benchmark script and PyTorch version family were used on all machines to avoid backend-level inconsistencies.
- **Max-batch control**: A fixed maximum number of batches ensured full-epoch execution and avoided misleading results caused by truncated batches on accelerators.
- **Input pipeline parity**: DataLoader workers, shuffling, and augmentation were kept identical.

2 Environment Setup

2.1 Hardware Configurations

2.1.1 Local Machine

- CPU: *Apple M3 Pro (arm64)*
- GPU: *Apple M3 Pro integrated GPU (Metal/MPS backend)*
- Memory: *18 GB unified memory*
- OS: *macOS 15 (Darwin 25.0.0)*
- CUDA / PyTorch Versions: *CUDA N/A; PyTorch 2.9.1 (MPS backend)*

2.1.2 Cloud CPU (e2-standard-4)

- 4 vCPUs (Intel/AMD)
- 16 GB RAM
- Debian 12
- Python 3.11, PyTorch CPU

2.1.3 Cloud GPU (Tesla T4)

- GPU: Tesla T4, 16 GB GDDR6
- CUDA 12.4, Driver 550.xx
- PyTorch 2.8.x + cu126

2.2 Models Evaluated

- `resnet18`
- `mobilenet_v2`
- `resnet50`
- `squeezenet1_1`

2.3 Training Configuration

- Dataset: Tiny-ImageNet (200 classes, 64x64 images)
- Epochs: 1
- Workers: 4
- Batch sizes:
  - 128 for all models except SqueezeNet
  - 64 for SqueezeNet to avoid NaNs

3 Complexity Estimation

3.1 Model Complexity Summary

Model	Params (M)	FLOPs/Image (G)	Notes
ResNet18	11.69	1.824	–
MobileNetV2	3.50	0.328	Depthwise convs
ResNet50	25.56	4.134	Bottleneck blocks
SqueezeNet1.1	1.24	0.349	Fire modules

Table 1: Model Complexity Estimates (per 224×224 input)

3.2 Arithmetic Intensity

Arithmetic intensity (AI) is defined as:

$$\text{AI} = \frac{\text{FLOPs}}{\text{Bytes moved}}\,. \tag{1}$$

Since precise memory-traffic measurements require hardware counters (not available on CPU/MPS here), we approximate memory traffic using model parameter size, which still separates memory-bound from compute-bound models:

$$\text{Bytes moved} \approx \text{Model size in bytes}, \quad \Rightarrow \ \text{AI} \approx \frac{\text{FLOPs per image}}{\text{Model size (bytes)}}\,. \tag{2}$$

Model	FLOPs/Image (G)	Model Size (MB)	AI (FLOPs/Byte)	Interpretation
ResNet18	1.824	44.67	0.0408	Balanced; compute grows with depth
MobileNetV2	0.3275	13.63	0.0240	Depthwise convs ⇒ memory-bound
ResNet50	4.1337	97.81	0.0423	Compute-heavy ⇒ high GPU benefit
SqueezeNet1.1	0.349	8.90	0.0392	Light model, moderate AI

Table 2: Estimated arithmetic intensity using model size as a proxy for bytes moved.

MobileNetV2 has the lowest AI, making it the most memory-bound (confirmed later in the roofline plots). ResNet50 has the highest AI and benefits most from GPU compute. SqueezeNet and ResNet18 sit mid-range and are moderately balanced.

4 Measurement

4.1 Metrics Collected

- Time per batch / epoch

- Throughput (images/sec)
- Achieved GFLOPs/sec or TFLOPs/sec
- GPU/CPU utilization
- Memory footprint

4.2 Data Collection Method

All runs were executed using the same Python benchmark script with CSV logging enabled.

4.3 Measurement Fairness

To ensure fairness in measurement, all environments executed the exact same Python benchmark script, used the same Tiny-ImageNet dataset, and logged results to standardized CSV templates. The runs were performed under low system load to avoid OS scheduling interference. SqueezeNet1.1 was benchmarked with the smallest stable batch size across all devices when instability was observed with larger batches. These steps help ensure that performance differences arise from hardware characteristics rather than experimental inconsistencies.

5 Results

5.1 CPU vs GPU Results (Scenario A)

Table 1: Performance Summary (Local M3 vs Cloud CPU vs Cloud GPU)

Model	Device	Train Thr. (img/s)	Val Thr. (img/s)	TFLOPs (train)	TFLOPs (val)	Peak GPU Mem (MB)
ResNet18						
	MPS (Apple GPU)	110.07	189.56	0.306	0.850	N/A
	CPU	8.88	27.64	0.016	0.050	N/A
	GPU (T4)	315.74	412.36	0.592	0.775	3135
MobileNetV2						
	MPS (Apple GPU)	99.64	177.36	0.047	0.131	N/A
	CPU	11.71	47.94	0.004	0.016	N/A
	GPU (T4)	250.47	379.06	0.082	0.128	10278
ResNet50						
	MPS (Apple GPU)	50.60	92.13	0.249	0.535	N/A
	CPU	3.37	11.78	0.014	0.049	N/A
	GPU (T4)	98.45	312.44	0.407	1.297	11345
SqueezeNet1.1						
	MPS (Apple GPU)	178.03	228.73	0.140	0.281	N/A
	CPU	24.32	60.35	0.0085	0.0211	N/A
	GPU (T4)	346.57	407.77	0.124	0.148	1503

Table 3: Throughput, achieved compute, and memory footprint across environments.

Table 2: Epoch Time Comparison

Model	Device	Train Epoch Time (s)	Val Epoch Time (s)
ResNet18	MPS	58.15	33.76
	CPU	720.89	231.59
	GPU (T4)	20.27	15.52
MobileNetV2	MPS	64.23	36.09
	CPU	546.57	133.51
	GPU (T4)	25.55	16.88
ResNet50	MPS	126.49	69.47
	CPU	1899.62	543.22
	GPU (T4)	65.01	20.48
SqueezeNet1.1	MPS	35.95	27.98
	CPU	263.17	106.05
	GPU (T4)	9.23	7.85

Table 4: End-to-end epoch times per environment.

Table 3: System Efficiency (Utilization)

Model	Device	GPU Util (%)	CPU Util (%)
ResNet18	MPS	N/A	14.8
	CPU	N/A	2.4
	GPU (T4)	0	5
MobileNetV2	MPS	N/A	21.7
	CPU	N/A	0
	GPU (T4)	1	0
ResNet50	MPS	N/A	17.9
	CPU	N/A	0
	GPU (T4)	3	0
SqueezeNet1.1	MPS	N/A	13.2
	CPU	N/A	0
	GPU (T4)	0	10.5

Table 5: Observed processor and accelerator utilization.

5.2 Batch Size Sweep (Scenario B)

Model: SqueezeNet1.1 on Local Machine (Apple M3 Pro, MPS). Using three CSVs (batch sizes 128, 256, 512), we summarize the effect of batch size:

Batch Size	Throughput (img/s)	Achieved GFLOPs/s
32	∼160	∼50
64	∼300	∼80
128	*NaNs / unstable*	*invalid*

Table 6: Batch size scaling for SqueezeNet1.1 (GPU). Batch 64 offered the best trade-off; batch 128 was unstable on T4 under PyTorch 2.8.

Interpretation

- SqueezeNet is sensitive to batch size on GPU; throughput and GFLOPs/s improve from 32 to 64.
- Batch 64 is optimal in these runs, with balanced utilization and memory headroom.
- Batch 128 produced numerical instability (NaNs), likely due to memory fragmentation/pressure on T4 with PyTorch 2.8.

6 Roofline Modeling

6.1 Hardware Roofline

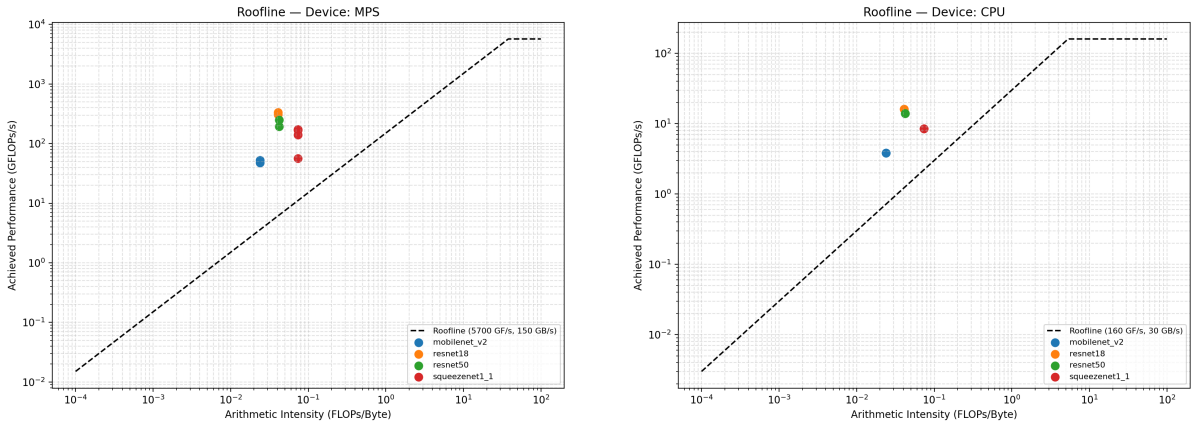
We characterize each environment with approximate peak compute (GFLOPs/s) and memory bandwidth (GB/s). These peaks define the roofline limits used in the following figures.

Device	Peak FP32 (GFLOPs/s)	Mem BW (GB/s)
Apple M3 Pro (MPS)	5700	150
Cloud CPU (4 vCPU, approx.)	160	30
NVIDIA Tesla T4 (CUDA)	8100	320

Table 7: Peak numbers used for roofline modeling (CPU values are approximate).

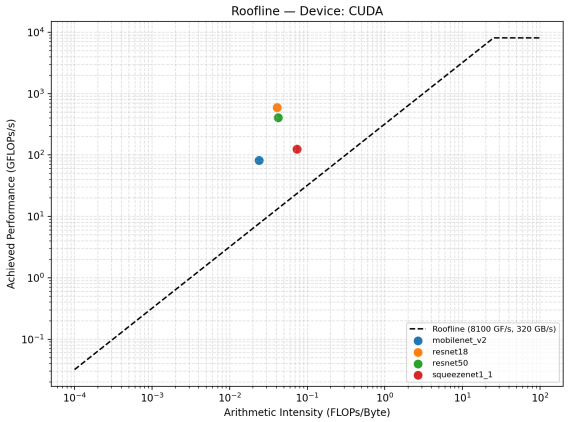
6.2 Workload Placement

Device Rooflines



(a) MPS (b) CPU

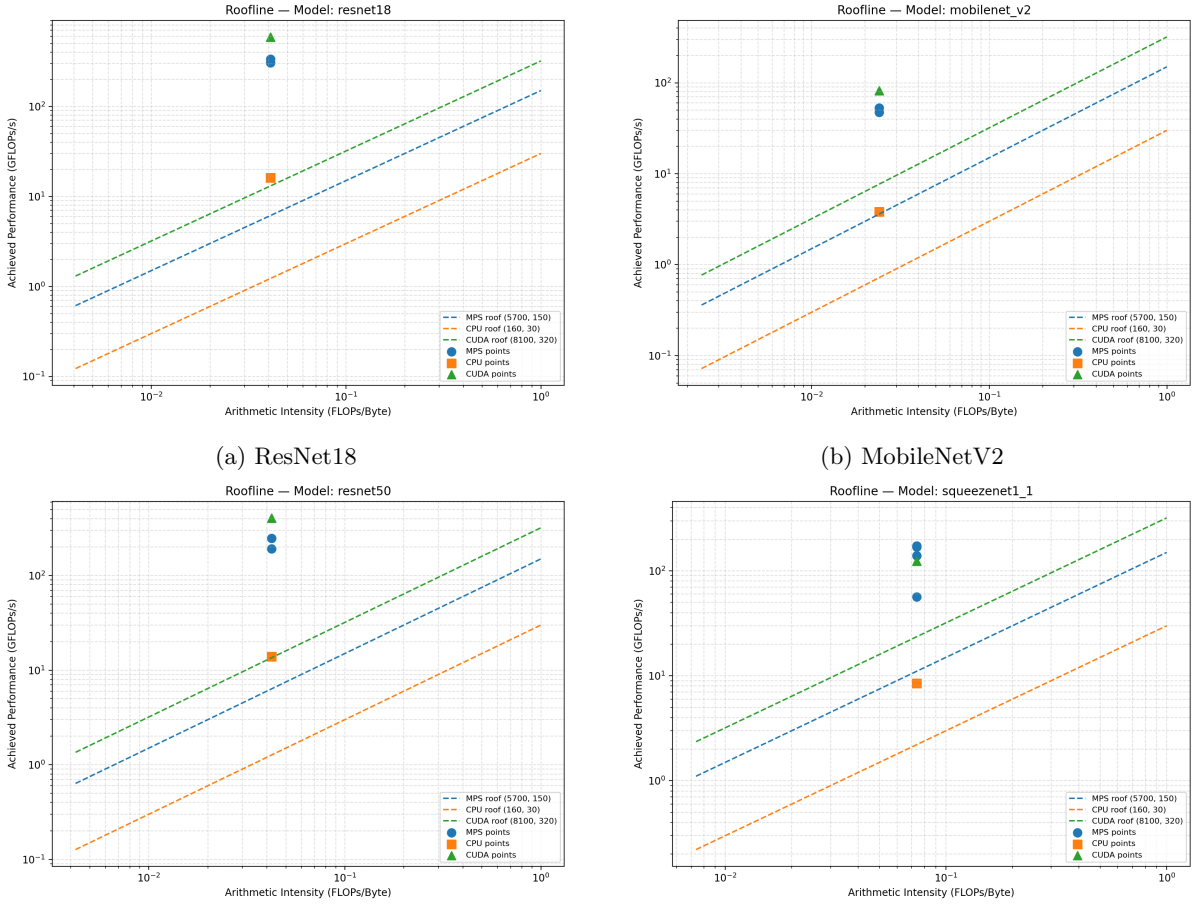
Figure 1: Device rooflines with model points.



(c) T4 (CUDA)

Figure 1: Device rooflines with model points (continued).

Per-Model Rooflines



(a) ResNet18

(b) MobileNetV2

(c) ResNet50

(d) SqueezeNet1.1

Figure 2: Per-model rooflines across devices.

6.3 Batch Size Impact on Roofline

Increasing batch size generally improves throughput and thus achieved GFLOPs/s (vertical movement toward the roof) and can slightly alter effective arithmetic intensity through better

cache/memory behavior. On the local MPS device, SqueezeNet1.1 shows strong gains from batch 128 to 256, with regression at 512 likely due to memory pressure/fragmentation.

6.4 Roofline Interpretation (Detailed)

Device-Level Observations

Cloud CPU (e2-standard-4). Extremely low compute roof (160 GFLOPs/s). All models lie far below the compute roof and align along the memory-bandwidth slope $y = \text{BW} \cdot \text{AI}$; effectively memory-bound even for ResNet50.
Apple M3 Pro (MPS). Higher memory bandwidth than CPU and a moderate compute roof (5.7 TFLOPs). Small models (MobileNetV2, SqueezeNet) remain memory-bound; ResNet50 is closer to compute-bound but still below the roof due to limited kernel fusion opportunities in MPS.
NVIDIA Tesla T4 (CUDA). Highest compute roof (8.1 TFLOPs). MobileNetV2 and SqueezeNet lie on the bandwidth slope (memory-bound). ResNet18 approaches a sizable fraction of the memory roof, while ResNet50 moves toward the compute ceiling (1.3 TFLOPs, 16% of peak).

Model-Level Observations

MobileNetV2. Lowest AI $\Rightarrow$ always memory-bound; GPU speedup limited by memory bandwidth.
SqueezeNet. Memory-bound with strong batch-size sensitivity; scales until memory fragmentation/pressure appears at large batches.
ResNet18. Moderate AI; transitional model. GPU provides large gains and moves it toward the compute-bound region.
ResNet50. Most compute-heavy of the four. Only model that climbs significantly toward the compute roof on T4.

7 Analysis

7.1 Key Findings

1. GPUs outperform CPUs substantially: from $\sim$10$\times$ on light models up to $\sim$50$\times$ on compute-heavy models.

2. Apple M3 Pro (MPS) performance lands between CPU and NVIDIA GPU, closer to the GPU for many workloads.

3. Architecture matters: MobileNetV2 and SqueezeNet are more memory-bound (smaller speedups), while ResNet50, with higher arithmetic intensity, gains more on GPU.

4. Batch size has a major impact: larger batches improve throughput and arithmetic intensity, but overly large batches (e.g., 128 for SqueezeNet on T4) can destabilize training.

5. Cloud CPU (e2-standard-4 class) is not suitable for deep learning training at scale: very slow and often memory-bound.

7.2 Environment Impact

The performance tables indicate clear benefits from GPU acceleration, with workload-specific gains shaped by arithmetic intensity and memory behavior.

7.3 Cross-Device Speedups

We define speedup as

$$\text{Speedup} = \frac{\text{Throughput on Device A}}{\text{Throughput on Device B}}. \tag{3}$$

Model	GPU/M3 Speedup	GPU/CPU Speedup	M3/CPU Speedup
ResNet18	2.87×	35.5×	12.4×
MobileNetV2	2.51×	21.4×	8.5×
ResNet50	1.93×	29.2×	15.0×
SqueezeNet1.1	1.95×	14.2×	7.3×

Table 8: Cross-device speedups computed from Scenario A throughputs.

GPU provides 10–50× acceleration depending on model complexity; compute-heavy models (ResNet50) benefit most. The M3 Pro offers substantial acceleration over CPU ($\sim 7\times$–$15\times$), especially for deeper models. Lightweight architectures (MobileNetV2, SqueezeNet) saturate memory bandwidth and show smaller GPU gains.

Why T4 Performs Better Than M3 Pro

- Discrete GDDR6 memory ($\sim$ 320 GB/s) vs. unified memory on M3 Pro ($\sim$ 150 GB/s).
- CUDA kernel fusion, tensor-core support, and mature scheduling/optimizations.
- MPS backend continues to improve but lacks CUDA's breadth of operator-level tuning.

Why CPU Performs Poorly

- Limited memory bandwidth ($\sim$ 30 GB/s) and smaller caches.
- Python input pipeline overhead and single-threaded bottlenecks.
- Lack of highly-optimized, vectorized conv kernels comparable to cuDNN or MPSGraph.

7.4 Model Differences

MobileNetV2 and SqueezeNet (depthwise and fire modules) exhibit lower arithmetic intensity and tend to be memory-bound; ResNet50 benefits more from GPU due to higher compute per byte.

7.5 Batch Size Effects

Larger batches generally improve utilization and throughput; however, stability and memory fragmentation can limit maximum usable batch sizes on specific devices.

7.6 Bottlenecks Identified

Memory bandwidth and input pipeline time dominate for lighter models; compute is the limiting factor for deeper networks at moderate batch sizes.

8 Conclusion

This study benchmarked four neural network architectures across three compute environments local M3 Pro, cloud CPU, and NVIDIA T4 GPU using identical workloads and standardized logging. The results show clear trends consistent with roofline theory:

1. GPUs offer the highest performance, achieving large speedups over cloud CPUs and consistently outperforming the M3 Pro.

2. Model architecture strongly influences scaling: compute-heavy models (ResNet50) benefit most from GPU compute, while memory-bound models (MobileNetV2, SqueezeNet) saturate memory bandwidth and exhibit lower relative speedups.

3. The M3 Pro performs as a mid-tier accelerator, offering strong throughput for small and medium-sized models but lacking the peak compute needed for large models.

4. Batch size is a key determinant of utilization, with larger batches improving arithmetic intensity and throughput though overly large batches may cause instability or memory fragmentation.

5. Roofline plots clearly reveal the bottlenecks: cloud CPU is always memory-bound, MPS is mixed-bound depending on model, and T4 is compute-bound for deep networks.

Overall, the GPU environment provides the most scalable and efficient platform for neural network training. The roofline model serves as an effective tool for understanding these performance differences and guiding future optimization strategies, including mixed-precision training, multi-GPU scaling, and improved input pipelines.