Performance Profiling of Neural Network Training

Tai Wan Kim

I. Introduction

As models get larger and training more expensive, performance has now become a necessity. Performance optimization poses a challenge, as it is effective when grounded in evidence, not guesswork. Here, we profile neural network training performance and explore what the numbers tell us.

We run targeted experiments across models and GPUs, focusing on a short, single training step rather than full runs. The experiments highlight two themes that go hand in hand: choosing models that utilize the hardware, and choosing hardware that eases the workload's bottleneck. Together, our results show how careful design translates to performance gains. Code and collected metrics have been included in the submission for review [code].

II. Experiment Design

We conduct a series of experiments aimed at observing the influence of model and GPU choices on training performance. We test two models: ResNet18, ResNet34; and two GPU flavors: T4 and L4 [1]. NYU Greene's Burst servers are used for all experiments (n1s8-t4-1, g2-standard-12).

Before we can formulate hypotheses, each GPU's peak compute P^{max} and bandwidth BW^{max} should be known. Note that both GPUs have the same BW^{max} while L4 has higher P^{max}.

GPU	P^{max} (FP32)	BW^{max}
T4 [2]	8.1 TFLOP/s	300 GB/s
L4 [3]	30.3 TFLOP/s	300 GB/s

We conduct three experiments:

Experiment A: Batch sweep on the same model and GPU We profile a training step of ResNet18 on L4 with batch size $B \in \{64, 256\}$. Increasing batch size will increase total FLOP/step and should lead to higher arithmetic intensity, since we reuse the same weights across more images. We expect the workload to move closer to the compute-bound side of the roofline, even if it may still remain left of the knee.

Experiment B: Cross-GPU at fixed model and batch sizes We profile a training step of ResNet18 on T4 and L4, with batch size fixed at 128. With the same model and batch size, total FLOP/step should be nearly identical on both GPUs, but L4 should complete the step faster and achieve higher GFLOP/s due to its higher peak compute and bandwidth. Since L4 has higher performance and a more sophisticated memory system, we also expect it to achieve a better AI.

Experiment C: Cross-model at fixed GPU & batch size We profile a training step of ResNet18 and ResNet34 on L4, with batch size fixed at 128. Switching from RN18 to RN34 will increase

both FLOPs and DRAM bytes per step, but FLOPs can grow faster if memory is reused effectively, so RN34 can have a higher AI. This should make RN34 relatively more compute-intensive (or less memory-bound) than RN18 on the roofline plot.

For all experiments, we use an NVTX range to profile exactly one step (forward + backward + optimizer). For consistent accounting, we disable automatic mixed precision and TensorFloat-32. We fix pre-profiling warmup to 20 iterations. CIFAR-10 dataset is used for all runs with the same input pipeline (identical transformation/augmentation applied) [4].

III. Complexity Analysis

i. Time Complexity Analysis

A time complexity analysis (Big-O) gives us a theoretical ground in understanding how cost scales (irrespective of hardware). The time complexity of typical Convolutional Neural Network layers is as follows:

$$T_{conv2D} = \Theta(B \times H_o \times W_o \times C_{out} \times (K^2 \times C_{in}/g))$$

$$T_{Pool} = \Theta(B \times H_o \times W_o\ C \times K^2)$$

$$T_{FC} = \Theta(B \times N_{in} \times N_{out})$$

B = Batch Size
H_o = Output Height (varies by layer/stage)
W_o = Output Width (varies by layer/stage)
C = Channels
C_{out} = Output Channels (varies by layer/stage)
C_{in} = Input Channels (varies by layer/stage)
K^2 = Kernel Size
g = group (1 in standard conv)
N_{in} = number of input neurons
N_{out} = number of output neurons

In CNNs, convolution has the highest order and dominates other operations. A training step is a constant-factor multiple of forward, typically approximated as ~×3 the forward [5]. For our experiments, we keep in mind that cost increases with B and the number of Conv2D layers in the model.

ii. Computational Cost Analysis (# FLOPs/step)

We also estimate the concrete per-step cost that can be compared to profiled values. The computational cost (number of FLOPs) of a forward pass on ResNet18 and ResNet34 are known to be 1.8 and 3.6 GFLOPs per image [1]. However, the numbers were sampled from ImageNet (224 × 224) architectures; since our models are trained on CIFAR-10 (32 × 32), we scale accordingly. The scale factor is $(32/224)^2 \approx 0.0204$. Then, a forward pass takes 0.03673 GFLOPs on ResNet18, and 0.07347 GFLOPs on ResNet34.

Again, a training step can be roughly approximated as ×3 the forward pass, so per-step FLOPs is 0.11019 GFLOPs on ResNet18, and 0.22041 GFLOPs on ResNet34. Then, we estimate the total FLOPs for each experimental setup, taking batch size into account.

Model	GFLOP/step (per img)	B	GFLOP/step

RN18	0.11019	64	7.09
		128	14.10
		256	28.20
RN34	0.22041	128	28.21

IV. Measurement

i. Methods

All metrics are collected using NVIDIA Nsight Compute (NCU). See Appendix section I for the NCU profiler arguments. For each kernel, we target the following metrics:

F_{add}^{k} (smsp__sass_thread_inst_executed_op_fadd_pred_on.sum):
F_{mul}^{k} (smsp__sass_thread_inst_executed_op_fmul_pred_on.sum):
F_{fused}^{k} (smsp__sass_thread_inst_executed_op_ffma_pred_on.sum):
B_{read}^{k} (dram__bytes_read.sum):
B_{write}^{k} (dram__bytes_write.sum):
$Time^{k}$ (gpu__time_duration.sum):

After the metrics have been collected, we sum the values across kernels, and compute attained FLOP/s and Arithmetic Intensity (FLOP/byte). This gives us a single point <Attained FLOP/s, AI> under the roofline ceiling.

$$F_{add} = \sum F_{add}^{k}$$
$$F_{mul} = \sum F_{mul}^{k}$$
$$F_{fused} = \sum F_{fused}^{k}$$
$$B_{read} = \sum B_{read}^{k}$$
$$B_{write} = \sum B_{write}^{k}$$
$$Time = \sum Time^{k}$$

$$F_{total} = F_{add} + F_{mul} + 2 \times F_{fused}$$
$$B_{total} = B_{read} + B_{write}$$

$$\text{Attained FLOP/s} = F_{total} / Time$$
$$AI = (FLOP/B) = F_{total} / B_{total}$$

While not necessary for roofline modeling, we also collect $SM\%_{k}$ and $DRAM\%_{k}$. $SM\%_{k}$ (sm__throughput.avg.pct_of_peak_sustained_elapsed) is the per-kernel compute utilization of the SMs (Streaming Multiprocessors), reported as a percentage of GPU's peak compute. $DRAM\%_{k}$ (dram__throughput.avg.pct_of_peak_sustained_elapsed) is the per-kernel DRAM bandwidth utilization, as a percentage of GPU's peak bandwidth. $Time^{k}$ is used to aggregate $SM\%_{k}$ and $DRAM\%_{k}$ into time-weighted averages (SM% and DRAM%) representative of the full step.

ii. Results

Below are the results from our experiments. The full set of extracted csv files and summaries have been included in the submission for review.

ID	Model	GPU	B	Total GFLOP	Total DRAM (GB)	Total Time (ms)
A-1	RN18	L4	64	4.928	0.744	7.376
A-2	RN18	L4	256	19.31	0.934	10.688
B-1	RN18	T4	128	9.579	1.108	14.402
B-2	RN18	L4	128	10.38	0.8074	8.456
C-1	RN18	L4	128	10.38	0.8074	8.456
C-2	RN34	L4	128	24.08	1.541	16.422

ID	Model	GPU	B	SM%	DRAM%	Attained GFLOP/s	AI
A-1	RN18	L4	64	13.647	23.164	668.144	6.623475
A-2	RN18	L4	256	34.485	23.304	1806.408	20.672312
B-1	RN18	T4	128	30.027	19.022	665.121	8.644452
B-2	RN18	L4	128	21.306	23.373	1227.597	12.856014
C-1	RN18	L4	128	21.306	23.373	1227.597	12.856014
C-2	RN34	L4	128	28.386	15.558	1466.556	15.624057

We can compare the attained GLOP/step with the computational cost analysis from section III. While there are differences, gaps are expected. The fact that the profiled GFLOP/step scales linearly with the batch size is a positive sign that the profiler is working as intended. For example, <RN18, B=256> is roughly ×4 <RN18, B=64>.

Model	B	Theoretical GFLOP/step	Profiled GFLOP/step
RN18	64	7.09	4.928
	128	14.10	9.579 (from B-1) 10.38 (from B-2)
	256	28.20	19.31

RN34	128	28.21	24.08

V. Roofline Modeling

A roofline model is an insightful visualization of workload performance [6]. See Appendix section II for the roofline models.

VI. Analysis

Experiment A: Batch sweep on the same model and GPU As B increases from 64 to 256, GFLOP/step scales almost linearly (4.928 → 19.31), while DRAM traffic remains roughly the same (0.744 → 0.934 GB). Because of this, AI is much higher when B=256 (6.623475 → 20.672312); we have more FLOPs per byte. The higher AI allows the SMs to stay busier (SM%: 13.647 → 34.485), while DRAM% stays essentially constant (23.164 → 23.304). This happens because the number of model weights does not scale with batch size, so memory traffic does not change dramatically. We can characterize B=256 as more compute-intensive, though both workloads do not fully utilize the GPU and remain to the left of the roofline ceiling's knee.

Experiment B: Cross-GPU at fixed model and batch sizes For both GPUs, GFLOP/step is very close, which is expected since we use the same batch size and model. On L4, we get higher throughput (665.121 → 1227.597 GFLOP/s) while issuing less DRAM traffic (1.108 → 0.8074 GB); it does more FLOPs while moving fewer bytes, which explains the higher AI. This is presumably due to L4 having a better design, e.g., more efficient kernels and caching. While SM% slightly dips on L4 (30.0 → 21.3), this is because it has a higher compute peak, so a smaller percentage still corresponds to more absolute FLOP/s. Overall, we can say it may be overkill to use L4 for this workload, as the higher compute capability is not being fully utilized.

Experiment C: Cross-model at fixed GPU & batch size Switching from RN18 to RN34 increases total GFLOP/step from 10.38 to 24.08 GFLOP (≈2.3×), while DRAM traffic grows more slowly (0.8074 → 1.541 GB, ≈1.9×). As a result, AI increases from 12.856 to 15.624. RN34 is a larger model, so each step does more compute per byte of DRAM traffic. This shows up in other metrics: throughput improves (1227.6 → 1466.6 GFLOP/s), SM% rises (21.3 → 28.4), and DRAM% actually drops (23.37 → 15.56), suggesting that RN34 is relatively more compute-intensive. However, both points remain to the left of the roofline knee.

There are two main takeaways. First, when training a neural network, we should make every byte count—do as much useful work as possible for each byte moved from DRAM. In our experiments, increasing the batch size and using a larger model improved arithmetic intensity and throughput without dramatically increasing DRAM traffic, which is exactly this effect. In practice, this can mean leaning into reuse (larger batches or gradient accumulation), trimming unnecessary traffic (fusion where possible), and keeping data close to the GPU (pin memory, persistent workers, prefetch) so we aren't stalled by I/O.

The second takeaway is to choose the right hardware. L4 is designed for compute-intensive workloads, where extra compute and specialized units (e.g., tensor cores, video engines) can be driven close to their peak. In our case, the workload is relatively small and low-intensity, so the higher compute

capability of L4 is not fully utilized and the step remains in the “memory-bound” region of the roofline. The takeaway is that the most expensive GPU may not always be the best choice; we should pick hardware whose strengths match the workload.

VII. Conclusion

We profiled a single-step training across models and GPUs and showed how model and hardware choices translate into performance gains. Along the way, we saw the value of Nsight Compute and roofline modeling for collecting and interpreting measurements. Two themes emerged: the memory wall and the need to choose hardware for the bottleneck.

For future work, we can explore with Nsight Compute for a more detailed and comprehensive view of training, and expand the matrix to include larger models (ResNet50) and higher-bandwidth GPUs (A100). This will help validate the trends we observed at scale, and test how far we can push utilization further when bandwidth isn’t the ceiling.

VIII. References

1. He, K., Zhang, X., Ren, S., and Sun, J., "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. doi:10.1109/CVPR.2016.90.
2. NVIDIA Corporation, "NVIDIA T4 Tensor Core GPU — Data Sheet."
3. NVIDIA Corporation, "NVIDIA L4 Tensor Core GPU — Data Sheet."
4. A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," Tech. Rep., University of Toronto, 2009.
5. DeepSpeed Team, "Flops Profiler," *DeepSpeed Tutorials*. Accessed: Nov. 11, 2025. Available: https://www.deepspeed.ai/tutorials/flops-profiler/
6. S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009. doi:10.1145/1498765.1498785.

IX. Appendix

i. NCU Profiler/Training Arguments

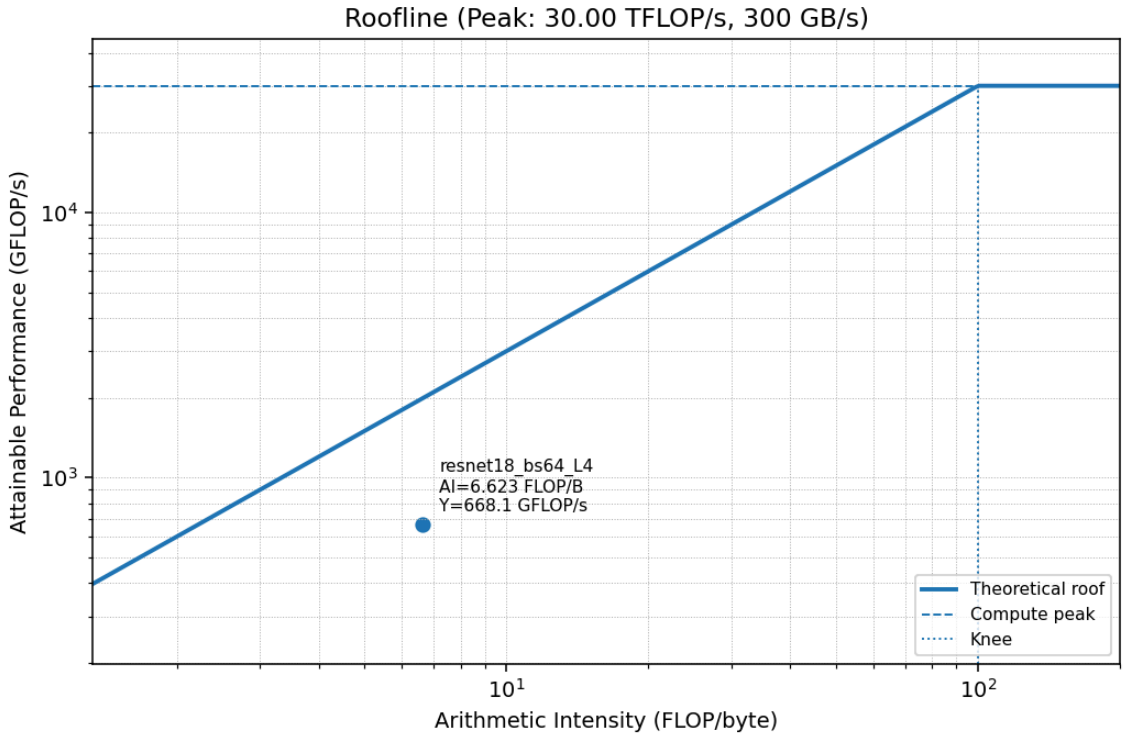
For reproducibility, I include the set of arguments I used for profiling.

NCU	--target-processes	all
	--metrics	smsp__sass_thread_inst_executed_op_fadd_pred_on.sum, smsp__sass_thread_inst_executed_op_fmul_pred_on.sum, smsp__sass_thread_inst_executed_op_ffma_pred_on.sum, dram__bytes_read.sum, dram__bytes_write.sum, gpu__time_duration.sum, sm__throughput.avg.pct_of_peak_sustained_elapsed, dram__throughput.avg.pct_of_peak_sustained_elapsed
Training	--profile-one-step	
	--warmup-iters	20

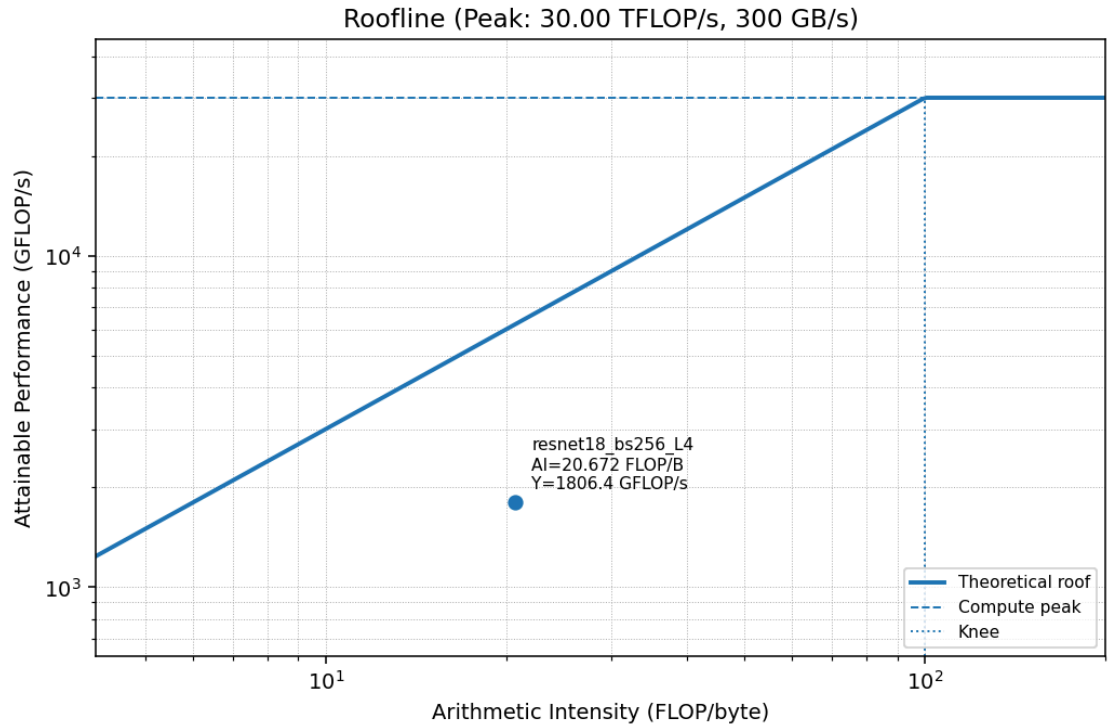
ii. Roofline Models

Note that the axes are in log scale.

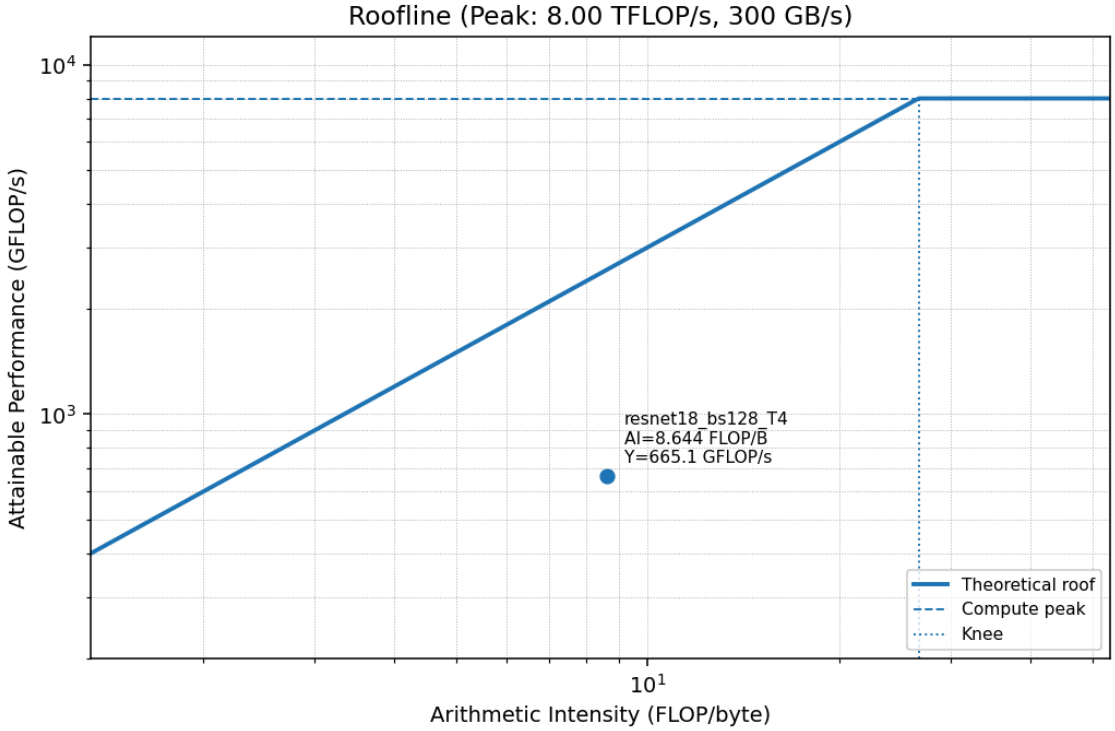
❖ Experiment A-1: RN18, L4, B=64



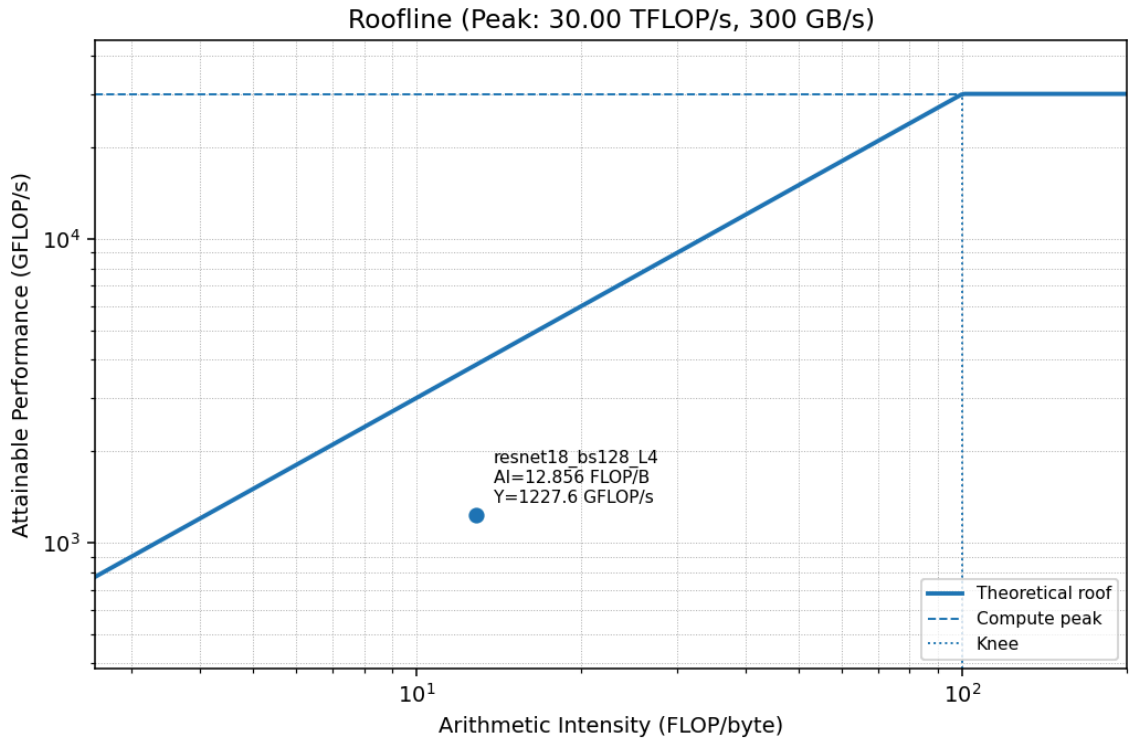
❖ Experiment A-2: RN18, L4, B= 256



❖ Experiment B-1: RN18, T4, B=128



❖ Experiment B-2/C-1: RN18, L4, B=128



❖ Experiment C-2: RN34, L4, B=128