Scalable Training: Pipeline vs Data Parallelism for ResNet and Vision Transformers

Ivan Čabrilo
University of Rochester
icabrilo@u.rochester.edu

Abstract

Large Machine Learning Models are computationally expensive to train, even on modern GPUs. As deep learning models continue to increase in size, choosing an effective approach to train them is becoming a customary fundamental of Machine Learning and broader field of AI Applications. In order to permit efficient training of such models, many methods for workload distribution are invented. Most notable approaches are model and data parallelism (as well as hybrids of these methods) which are becoming increasingly crucial for achieving high throughput while preserving convergence behavior. This project empirically evaluates and compares model parallelism with data parallelism, through two experimental components. First, we evaluate pipeline parallelism on deep CNN architectures (ResNet-18/34/50/152), systematically varying global batch size and microbatch count to study throughput and convergence. Second, we compare model parallelism (pipeline parallelism) with data parallelism on a large ViT-L/16 and H/14 models (300M & 600M parameters). Preliminary results show that pipeline parallelism enables training of larger models, whereas data parallelism enables higher throughput. Presented findings provide guidance for selecting parallelization methods based on model size and accessible hardware.

1. Introduction

Training deep neural networks increasingly requires multi-GPU computation. While modern frameworks provide several parallelization primitives, selecting a suitable strategy depends heavily on model architecture and depth. In this work, we investigated the practical trade offs between Data and Pipeline parallelism.

Although data parallelism is the dominant paradigm in practice, it suffers from gradient synchronization overhead and memory redundancy. Pipeline parallelism provides an alternative by reducing memory usage and increasing model capacity per GPU, at the cost of pipeline bubbles and schedule complexity [5]. Prior work such as GPipe [3], PipeDream [2], and DeepSpeed pipeline engine [6] demonstrates the potential of PP for extremely large models, yet its behavior on intermediate depth CNNs and CNN based transformers is not thoroughly documented.

This project aims to explore optimal batch size and number of mini batches for PP, trained across ResNet-18, 34, 50, and 152 using identical training conditions. Beyond simple throughput measurements, an important goal of this work is to understand how these parallelization strategies behave in realistic, slightly messy settings rather than idealized ones. In practice, engineers rarely operate under perfectly tuned hyperparameters, perfectly balanced model partitions, or theoretically optimal scheduling policies. Instead, one often starts from a reasonable implementation, observes where it breaks (due to OOM errors, numerical stability, or GPU under utilization), and and iteratively refines the system. By combining quantitative measurements with qualitative observations from this debugging process, the project aims to provide not only raw performance numbers, but also practical guidance on how to reason about and diagnose common failure modes in scalable training.

Moreover, the comparison between CNN-based architectures (ResNets) and transformer based architectures (ViTs) allowed us to study two very different kinds of workloads on the same hardware. ResNets trained on CIFAR-10 represent a relatively lightweight, high throughput regime where individual forward passes are cheap, while ViT-L/16 and ViT-H/14 represent heavy, memory bound models closer to what one might see in large scale vision or multimodal systems. This contrast helped clarify in which situation pipeline parallelism is merely an overcomplication and in which it is effectively a prerequisite for training the model at all.

Contributions:

1. A systematic throughput and scaling analysis of ResNet architectures across pipeline parallel.

2. An empirical study of microbatching and its effect on pipeline bubble reduction.
3. A practical performance characterization of multi-GPU training on RTX A5000 hardware, contributing reproducible insights.

2. Background

2.1. Parallel Training Strategies

Data Parallelism (DP). DP replicates the full model on each GPU and synchronizes gradients after backpropagation via all-reduce [4]. It is simple and effective but suffers from:

- increased synchronization overhead for large models,
- redundant memory usage,
- suboptimal scaling when communication dominates computation.

Pipeline Parallelism (PP). Pipeline parallelism: is a specific type of model parallelism. Model is cut into stages (usually couple of NN layers per stage) and each stage lives on a different GPU. The input batch is further broken into micro batches that flow through the stages. While stage one works on micro batch three stage two is already on micro batch two and so on.

```
time 1: S1(m1)
time 2: S1(m2) S2(m1)
time 3: S1(m3) S2(m2) S3(m1)
time 4: S1(m4) S2(m3) S3(m2)
time 5:        S2(m4) S3(m3)
time 6:               S3(m4)
```

Useful when the network is deep and stages can be balanced by memory and compute. Works such as GPipe [3] proposes schedules to mitigate pipeline bubbles, the primary source of inefficiency.

PP is attractive for:

- extremely deep architectures,
- memory constrained environments,
- model parallel workloads where DP alone is insufficient.

2.2. Related Work

Prior studies primarily focused on:

- GPipe applied to deep CNNs and ResNets [3];
- Activation checkpointing for memory efficiency [1];
- Pipeline schedules with reduced stalls [5].

In addition to these methods, there has been increasing interest in hybrid strategies that combine data, tensor and pipeline parallelism within a single training job. Large language model training systems often deploy some form of multi dimensional parallelism, where model parallelism handles intra layer matrix multiplications, pipeline parallelism splits the model depthwise, and data parallelism replicates the resulting composite model across nodes. While reproducing such setups is beyond the scope of this project, the fundamental trade offs are shared: communication bandwidth versus memory footprint, pipeline bubbles versus utilization, and numerical stability under mixed precision training. Additionally, our work aims to provide complementary experimental insight for mid scale clusters, as well as comparison between data and model parallelism on 300 & 600 million parameter models.

3. Method

We evaluted:

- **PP on 2 and 4 GPUs with varying numbers and sizes of microbatches**
- **DP vs. PP comparison on ResNet-152 and ViT-L/16 & H/14 models**

For the first part (ResNet-18/34/50), all models were trained using SGD with momentum, global batch sizes ranging from 128–2048, and microbatch counts $m \in \{4, 8\}$, which gives 10 different configurations per model.

Therefore, we trained same model twice, once with 2 GPUs and once with 4 GPUs:

```
2gpu_train_pipeline_resnet(18/34/50).py
4gpu_train_pipeline_resnet(18/34/50).py
```

For our comparative analysis, we trained ResNet-152 using both DP and PP on 2 GPUs for 80 epochs, followed by ViT-L/16 with DP and ViT-H/14 with PP on 4 GPUs for 10 epochs each.

3.1. Datasets

We use **CIFAR-10** (50k training, 10k test images) to train all of our models. ResNet models operate directly on the original 32×32 inputs. For ViT models, CIFAR-10 images are upsampled to 224×224, since the model expects 224×224 resolution with 16×16 patch size.

3.2. Implementation Details

All experiments were implemented in PyTorch using the `torchvision` model zoo for ResNets and publicly available ViT implementations as starting points. For ResNet experiments, we used stochastic gradient descent (SGD) with momentum 0.9 and weight decay of 5×10^{-4}. The initial learning rate was chosen based on the global batch size, following a simple linear scaling rule, and decayed using a cosine schedule over the training epochs. For ViT models, we used AdamW with standard transformer style hyperparameters and enabled Automatic Mixed Precision (AMP) to keep memory usage within the 24 GB limit of the RTX A5000 GPUs.

Data preprocessing for CIFAR-10 included per channel normalization and standard augmentation. No aggressive regularization techniques (such as Mixup or CutMix) were applied, as the focus of the work was on parallelization behavior rather than achieving state of the art accuracy. All runs used shuffled minibatches and synchronized random seeds per configuration to make comparisons between DP and PP easier to interpret.

For data parallelism, we used DistributedDataParallel (DDP) in a single GPU configuration. Gradient synchronization was performed at every step. For pipeline parallelism, we initially relied on higher level pipeline abstractions to partition models into stages and route microbatches between devices. This initial implementation turned out to be a major source of both poor performance and numerical instability for ViT-H/14, motivating a second, more explicit implementation described later in more detail.

4. Experiments

Experimental evaluation is organized into two main parts. First, we studied ResNet-18/34/50/152 on CIFAR-10 to analyze how pipeline parallelism behaves for moderately deep convolutional networks at relatively low input resolutions. In this regime, the models can already fit on a single GPU, so PP is not strictly required for memory reasons, and instead, we recorded most optimal batch size and microbatch count. Furthermore we compared pipeline parallelism with data parallelism on throughput and scaling efficiency when it comes to ResNet-152 model. Second, we inspected ViT-L/16 and ViT-H/14, which are significantly larger and closer to the memory limits of the hardware. Here, pipeline parallelism becomes important as an enabling tool that allowed training models that data parallelism alone could not handle. For each configuration, we measured:

- **Throughput**, reported as images processed per second, averaged over training epochs,
- **Time per epoch**, which reflects both GPU utilization and communication overheads,

All throughput numbers were obtained on the same 4×A5000 node, using identical software versions and batch size settings for fair comparison between DP and PP. When an experiment crashed due to out of memory (OOM) or NaNs, we recorded this as a failure mode rather than silently discarding the run, since these failures convey useful information about the practical limits of each parallelization strategy as well as poor implementation.

4.1. Throughput Results

For a pipeline with n stages and m microbatches, the ideal utilization is given by

$$\text{Utilization} = \frac{m}{m+n-1}$$

With $n = 2$ stages and $m = 4$ microbatches, pipeline utalization is $\frac{4}{4+2-1} = 0.8$, yielding a theoretical speedup of $1.6\times$ on 2 GPUs over a single GPU. However, our empirical results with ResNet-18/34/50 reveal that practical performance deviates from this theoretical optimum due to communication overhead and stage imbalance. We found that smaller microbatch counts generally yielded better performance, with an optimal global batch size of 256 (microbatch size of 64) emerging across most configurations. While these findings are specific to ResNet architectures, they highlight key insight, pipeline parallelism doesn't produce perfect linear scaling.

```
res18  | 2 GPU | m=4 | Batch= 512 | Acc: 0.755 | Throughput: 26663.6 img/s
res18  | 4 GPU | m=4 | Batch= 256 | Acc: 0.753 | Throughput: 13886.0 img/s
res34  | 2 GPU | m=4 | Batch= 256 | Acc: 0.731 | Throughput:  9993.5 img/s
res34  | 4 GPU | m=4 | Batch= 256 | Acc: 0.721 | Throughput:  9228.5 img/s
res50  | 2 GPU | m=4 | Batch= 256 | Acc: 0.598 | Throughput:  5907.4 img/s
res50  | 4 GPU | m=4 | Batch= 128 | Acc: 0.583 | Throughput:  3777.4 img/s
```

Figure 1. Best performing model set ups for pipeline parallelism

The throughput results for ResNet-18, ResNet-34, and ResNet-50 reveal three notable trends. First, throughput decreases as model depth increases, since deeper networks such as ResNet-34 and ResNet-50 require more computations per image, resulting in fewer processed images per second compared to the lighter ResNet-18. This illustrates the classic trade off between model complexity and speed: deeper models typically yield higher accuracy but at the cost of reduced throughput. Second, increasing the number of GPUs from 2 to 4 does not necessarily lead to proportional throughput gains. For instance, ResNet-18 achieves 26,663 img/s on 2 GPUs but only 13,886 img/s on 4 GPUs, representing an almost 50% drop. This result although seemingly counterintuitive is the product of pipeline bubble overheads and communication latencies, which become prominent when training shallow models on small CIFAR-10 images (32×32), where per microbatch computation is very limited. In other words, pipeline parallelism is not effective for models that can be trained on single GPUs. Since GPUs spend more time communicating than computing, and pipeline stages remain severely underutilized. Finally, accuracy remains relatively stable across GPU configurations, indicating that adding more GPUs affects only throughput and not convergence. These observations confirmed that pipeline parallelism is poorly suited for shallow architectures and small input resolutions, where the compute to communication ratio is too low to make up for overheads effectively. The reason we notice better accuracy in smaller models is because all models were trained on 15 epochs each, and larger models required more training epochs. A natural question is whether increasing the microbatch count m can compensate for these inefficiencies by reducing pipeline bubbles. In theory, larger m pushes the utilization $\frac{m}{m+n-1}$ closer to 1 as long as the stages remain busy. In practice, we found that moving from $m = 4$

to $m = 8$ did not yield substantial throughput gains on ResNet models. The per microbatch workload is so small on CIFAR-10 that the added communication and scheduling overheads dominate any marginal improvements from deeper pipelines. In some configurations, higher m even slightly degraded throughput, likely due to increased pressure on the CUDA stream scheduler and more frequent synchronization points.

This behavior highlights a key discovery: microbatch tuning is not a free knob that can always "fix" pipeline performance. When individual forward and backward passes are cheap, the overhead from managing many tiny microbatches quickly becomes visible, and the theoretical utilization model ceases to be predictive. For the ResNet regime considered here, moderate microbatch counts with a global batch size of 256 with $m = 4$ struck the best balance (on average) between utilization and overhead.

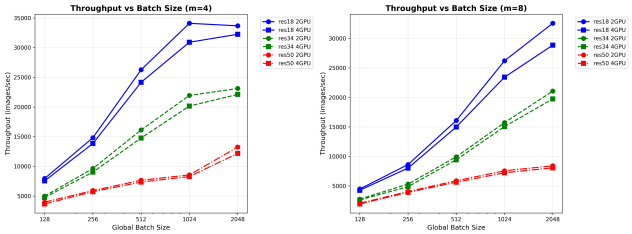

Figure 2. Throughput vs. batch size for PP across 2 and 4 GPU

4.2. Scaling Efficiency

Figure 3 illustrates the scaling efficiency when increasing the number of GPUs from 2 to 4 for three different ResNet architectures, with a fixed microbatch count ($m = 4$). Scaling efficiency is defined as the ratio of throughput achieved on 4 GPUs to that achieved on 2 GPUs.

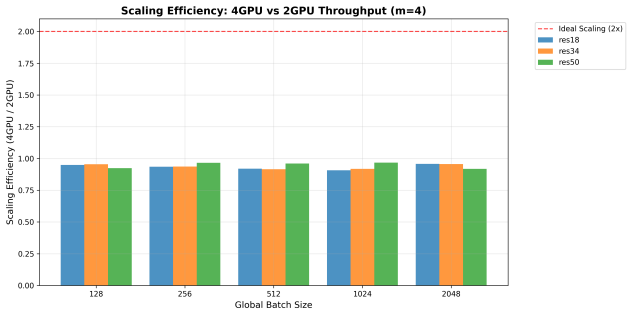

Figure 3. Scaling efficiency from 2 to 4 GPUs for different models

Ideal linear speedup would yield a value of 2.0 (indicated by the red dashed line). However, across all models and global batch sizes, we observe efficiencies consistently below 1.0, meaning that doubling the number of GPUs does not lead to proportional speedup. In fact, in all cases, throughput per GPU decreases when moving to 4 GPUs. Indicated by all values being below value of 1.

This inefficiency is attributed to fundamental limitations of pipeline parallelism for smaller models. The computational workload per microbatch—dictated by both model depth and CIFAR-10's 32×32 resolution—is insufficient to amortize pipeline overhead. Consequently, communication latency and synchronization costs dominate execution time, negating any benefits from additional GPUs. Even ResNet-50, while deeper, remains too computationally lightweight to overcome these bottlenecks, and performs better on 2 than 4 GPUs

4.3. Accuracy Curves

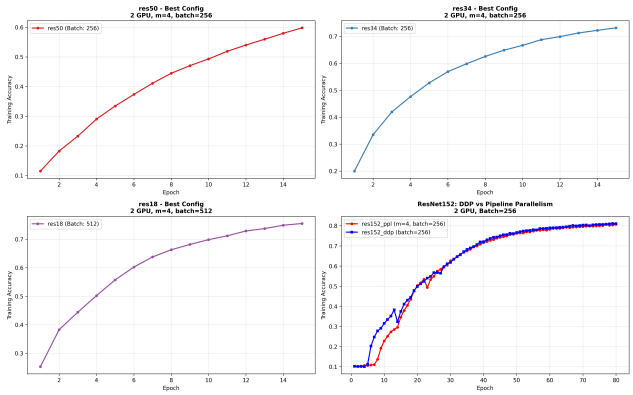

Figure 4. Training accuracy progression for each architecture's best configuration

The accuracy curves in Figure 4 show several important trends. First, shallower architectures such as ResNet-18 exhibit faster convergence, reaching over 70% accuracy within the first 10 epochs. This is expected due to their lower parameter count and reduced complexity, allowing for quicker optimization and smoother learning. In contrast, deeper models, ResNet-34 and ResNet-50 converge more gradually, requiring additional epochs to fully leverage their higher capacity.

On the last graph in Figure 4 we compare DP and PP on ResNet-152. Comparison reveals that both parallelization strategies achieve nearly identical convergence behavior, reaching approximately 80% final accuracy. This demonstrates that the choice of parallelism primarily affects training efficiency and execution rather than learning quality. Notably, PP exhibits a slightly noisier and slower start due to bubble overhead and delayed weight synchronization across pipeline stages, but once the pipeline is saturated, the convergence trajectories closely align with DP.

Furthermore Figure 5 shows that PP for ResNet-152 behaves the best with global batch size of 256 and 4 microbatches. Both DP and PP approaches were trained for 80

```
res152     | 2 GPU | m=4 | Batch= 256 | Acc: 0.807 | Throughput:  1856.4 img/s
res152_ddp | 2 GPU | m=1 | Batch= 256 | Acc: 0.810 | Throughput:  3891.5 img/s
```

Figure 5. ResNet-152 with DP and PP on 2 GPUs

epochs which shows that larger models need more epochs to converge.

These findings confirm that (i) deeper models require more training iterations to converge and that (ii) parallelization strategy does not significantly impact convergence performance, and instead focuses on efficiency and scalability, leading us to identify a global batch size of 256 with 4 microbatches as the optimal setup for PP in ResNets.

4.4. Vision Transformer Training

The second phase of our study demonstrates how and when to use which strategy. DP, although at its limits, successfully trained ViT-L/16. Whereas, PP enabled comfortable training of 2x larger ViT-H/14 model. Both configurations utilized Automatic Mixed Precision (AMP) to enhance memory efficiency.

4.5. GPU Utilization

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 580.105.08             Driver Version: 580.105.08     CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA RTX A5000               On  |   00000000:2A:00.0 Off |                    0 |
| 51%   78C    P2             223W /  230W |   20088MiB /  23028MiB |     100%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA RTX A5000               On  |   00000000:3D:00.0 Off |                    0 |
| 48%   76C    P2             226W /  230W |   20088MiB /  23028MiB |     100%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   2  NVIDIA RTX A5000               On  |   00000000:AB:00.0 Off |                    0 |
| 46%   73C    P2             225W /  230W |   20088MiB /  23028MiB |     100%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   3  NVIDIA RTX A5000               On  |   00000000:BD:00.0 Off |                    0 |
| 43%   72C    P2             223W /  230W |   20088MiB /  23028MiB |     100%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
```

Figure 6. GPU utilization under 4-GPU DP for ViT-L/16

Figure 6 shows memory usage of GPUs during training of ViT-L/16 with 300 million parameters, and global batch size of 192. Global batch size of 224 was attempted but torch.OutOfMemoryError: CUDA out of memory error was encountered, therefore this is the most usage we could squeeze out of our GPUs.

With DP training on N_{GPU} devices, the global batch B_{global} is split into per–GPU minibatches of size $B_{\text{local}} = B_{\text{global}}/N_{\text{GPU}}$. The memory footprint on a single GPU can be decomposed into a batch–independent part (parameters, optimizer state, gradients, static buffers) and a batch–dependent part coming from activations and their gradients. For mixed–precision Adam training this can be approximated as

$$M_{\text{DP}}(B_{\text{local}}) \;\approx\; M_{\text{model}} + k\, B_{\text{local}},$$

where M_{model} is the batch–independent footprint and k is the per–sample activation+gradient cost.

For ViT-L/16 the parameter count is approximately $N_{\text{param}} \approx 3 \times 10^8$. In mixed precision with Adam, each parameter is typically stored as: one FP16 weight, a master FP32 copy, two FP32 Adam moments, and an FP16 gradient. This corresponds to

$$\text{bytes/param} \approx 2 + 4 + 4 + 4 + 2 = 16 \text{ B},$$

so the batch–independent model+optimizer footprint is

$$M_{\text{model}} \approx \frac{N_{\text{param}} \cdot 16}{2^{20}} \approx \frac{3 \times 10^8 \cdot 16}{1,048,576} \approx 4,685 \text{ MiB}$$

Empirically, with 4 way DP and $B_{\text{global}} = 192$ ($B_{\text{local}} = 48$) each A5000 reports about $M_{\text{DP}}(48) \approx 20{,}088$ MiB of memory usage. Subtracting the static footprint gives an estimate of the activation cost at this batch size:

$$M_{\text{act}}(48) \approx M_{\text{DP}}(48) - M_{\text{model}} \approx 20{,}088 - 4,685 \approx 15{,}403 \text{ MiB},$$

so the per sample activation+gradient cost is

$$k \approx \frac{M_{\text{act}}(48)}{48} \approx \frac{15{,}403}{48} \approx 321 \text{ MiB / sample}.$$

At $B_{\text{local}} = 56$ (corresponding to $B_{\text{global}} = 224$) this linear model predicts

$$M_{\text{DP}}(56) \approx M_{\text{model}} + 56k \approx 4{,}578 + 56 \cdot 321 \approx 22{,}661 \text{ MiB},$$

This approximation is very close to the observed CUDA OOM error: "Of the allocated memory 20.71 GiB is allocated by PyTorch, and 803.96 MiB is reserved by PyTorch but unallocated", meaning that due fragmentation of PyTorch library, we could not use any extra space in memory to increase global batch size.

The key observation is that increasing B_{global} from 192 to 224 adds only eight samples per GPU, yet those eight samples cost roughly $8 \cdot 321 \approx 2.6$ GiB of extra activation memory on top of an already nearly full ≈ 20 GiB footprint. This explains why $B_{\text{global}} = 192$ is stable, while adding just 32 more samples at the global level (eight per GPU) immediately triggers an OOM. The remaining space for activations scales linearly with B_{local} and can be exhausted by a surprisingly small batch size change under DP.

For scaling under PP the model is partitioned into n stages placed on different GPUs. The memory on stage i can be written schematically as

$$M_{\text{PP},i} \;\approx\; M_{\text{model},i} \;+\; k_i\, \ell_i,$$

where $M_{\text{model},i}$ is the static footprint of the parameters and optimizer state residing on that stage, k_i is the per microbatch activation cost for that stage, and ℓ_i is the maximum

number of microbatches that are simultaneously "in flight" on that stage. Unlike DP, where every GPU holds the full model and B_{local} controls memory, here each GPU holds only a slice of the model but must retain activations for several microbatches at once until their backward passes complete.

In the warm portion of the pipeline (after the initial fill and before the final drain) each stage processes roughly one microbatch per time step. For a schedule with m microbatches and n stages, the maximum number of outstanding microbatches per stage is on the order of $\ell_i \approx \mathcal{O}(m)$, so the activation term effectively scales with both global batch size and the number of microbatches. Increasing m reduces pipeline bubbles and improves utilization, but also increases ℓ_i and therefore the peak activation memory on each device. In our ViT-H/14 experiments we observe that even though the model weights are spread across four GPUs, memory can still be exhausted if the product of global batch size and number of microbatches is too large, because each stage must buffer multiple 600M-parameter transformer activations concurrently. Thus, pipeline parallelism trades parameter memory for additional activation memory: it enables training of models that do not fit into a single GPU under DP, but it introduces a second memory axis controlled by the microbatch count m, which must be tuned jointly with B_{global} to avoid OOM errors.

ViT-H/14 with Pipeline Parallelism

For a given stage s of PP, the memory footprint can be written schematically as

$$M_{\text{PP}}^{(s)} \approx M_{\text{stage}}^{(s)} + C^{(s)} \cdot b_\mu \cdot N_{\text{active}}^{(s)},$$

where $M_{\text{stage}}^{(s)}$ is the parameter/optimizer footprint of that stage, $C^{(s)}$ is the per sample activation cost for its layers, b_μ microbatch size and $N_{\text{active}}^{(s)}$ is the number of microbatches for which stage s simultaneously holds activations.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 580.105.08             Driver Version: 580.105.08     CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA RTX A5000               On  |   00000000:2A:00.0 Off |                    0 |
| 38%   69C    P2             179W / 230W |   17058MiB /  23028MiB |     61%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA RTX A5000               On  |   00000000:3D:00.0 Off |                    0 |
| 34%   67C    P2             183W / 230W |   17040MiB /  23028MiB |     48%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   2  NVIDIA RTX A5000               On  |   00000000:AB:00.0 Off |                    0 |
| 35%   67C    P2             186W / 230W |   17040MiB /  23028MiB |     93%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   3  NVIDIA RTX A5000               On  |   00000000:BD:00.0 Off |                    0 |
| 30%   63C    P2             183W / 230W |   17022MiB /  23028MiB |     59%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
```

Figure 7. GPU utilization under 4-GPU pipeline parallelism for ViT-H/14

Two important consequences follow. First, increasing the *global* batch size B at fixed m increases the microbatch size b_μ and therefore linearly increases activation memory on *every* stage, exactly as in DP. This is why pipeline runs also hit CUDA OOM when B and m are chosen too aggressively: each stage still needs to keep forward activations for its in flight microbatches until their corresponding backward passes are computed. Second, increasing the number of microbatches m at fixed B decreases b_μ but increases the pipeline depth of in flight work, since $N_{\text{active}}^{(s)}$ grows, stages can end up holding activations for more microbatches at once, again increasing the memory term $C^{(s)} \cdot b_\mu \cdot N_{\text{active}}^{(s)}$.

In practice there is a three way trade off between global batch size, number of microbatches, and available VRAM. This trade off becomes particularly critical for large models like ViT-H/14. This fundamental constraint motivated our experimental design: we ran PP on larger model than DP to demonstrate PP's unique capability to train models that otherwise would not be.

Debugging ViT-H/14: NaNs and Underutilization

The final ViT-H/14 results reported in Table 1 were not obtained on the first attempt. In early experiments with the initial PP implementation, training exhibited two problematic symptoms. First, the scalar loss would output NaN for each iteration. Since the loss turned NaN, the optimizer state also became corrupted and the run had to be interrupted. Second, profiler traces and repeated `nvidia-smi` snapshots showed extremely unbalanced GPU utilization: one device would frequently sit around 90% utilization, while another remained effectively idle at 2%, with the remaining two GPUs hovering between 10% and 20%. Under these conditions, a single epoch of ViT-H/14 training took on the order of 1500 seconds.

The NaN issue was ultimately traced to gradient handling across microbatches. In the buggy version, gradients from multiple microbatches were accumulated without proper normalization before performing the optimizer step. This is equivalent to multiplying the effective learning rate by the number of microbatches, which is particularly problematic in combination with AMP and a very large parameter space ($\sim$600M parameters). Occasional large updates would push weights or activations into ranges where intermediate values overflowed, eventually producing NaNs in the loss.

At the same time, the skewed utilization pattern reflected an imbalanced stage partition and a suboptimal schedule. One stage contained a disproportionate share of the transformer blocks and dominated the critical path, while other stages frequently waited for work. In other words, the system had enough raw GPU capacity, but the pipeline was starved on some stages and overloaded on others, leading to both poor throughput and wasted hardware.

To address these problems, the pipeline implementation was rewritten to make stage boundaries, microbatching, and

gradient accumulation fully explicit. The model was repartitioned so that each of the four GPUs hosted a comparable number of transformer blocks and roughly similar memory footprints. During the forward pass, microbatches were sent through the stages in a fixed schedule designed to keep all GPUs busy after the pipeline warm up. During the backward pass, gradients were accumulated locally and then divided by the number of microbatches before calling the optimizer's `step()` function, restoring the intended effective learning rate.

After these changes, the training dynamics changed dramatically. Loss values remained finite throughout all ten epochs of ViT-H/14 training, and additional checks for NaNs or Infs in gradients did not fire. The GPU utilization plot (Figure 7) showed that all four A5000s now maintained nontrivial utilization over time, rather than the earlier "90–2–10–20%" pattern. Most importantly, the wall-clock time per epoch dropped from roughly 1500 seconds in the initial implementation to about 450 seconds in the final one, bringing PP performance into the range reported in Table 1. These observations underscore that, for large models, pipeline parallelism is highly sensitive to implementation details and that careful debugging can turn an almost unusable configuration into a practical one without changing the underlying hardware.

In our ResNet152 experiments, DP achieved twice the throughput of PP while maintaining same accuracy. As we can see that comparison is misleading for understanding PP's true value: training models that do not fit within single-GPU.

Table 1 and 2 show PP and DDP on models at their respective scalability limits. The ViT-H/14 model, shows consistent training progress despite lower absolute throughput, and slower convergence. Meanwhile, ViT-L/16 trained with DP achieves higher throughput and faster convergence, but represents the upper bound of what fits in single-GPU memory. This contrast highlights PP's role as an enabling training rather than a performance optimization.

Table 1. Training Progress: ViT-H/14 with Pipeline Parallelism (4 GPUs)

Epoch	Loss	Accuracy	Throughput	Time
1	2.2716	9.98%	108.53	459.95s
2	2.1833	10.38%	110.43	452.06s
3	2.0936	10.58%	110.41	452.11s
4	1.9936	10.67%	110.37	452.31s
5	1.9219	11.04%	110.37	452.30s
6	1.7672	10.95%	110.38	452.25s
7	1.6175	11.27%	110.39	452.22s
8	1.5937	11.74%	110.42	452.10s
9	1.5520	12.08%	110.49	451.81s
10	1.5132	13.27%	110.37	452.29s

Cumulative time: 4529.40s. The training exhibits several notable characteristics: throughput stabilizes around 110 samples/second after the first epoch, indicating somewhat efficient pipeline utilization. The consistent epoch times demonstrate stable pipeline balancing across all four GPUs (each GPU uses same amount of memory). Most importantly, the decrease in loss confirms that meaningful learning occurs despite the architectural complexity introduced by pipeline parallelism.

Warm up Behavior and Pipeline Saturation. A closer look at Table 1 reveals an important characteristic of PP training: the system exhibits a noticeable warm up period during the first epoch, after which throughput stabilizes. In epoch 1, the throughput is 108.53 img/s, noticeably lower than the steady-state values of roughly 110–110.5 img/s observed in epochs 2–10. The epoch time follows the same pattern, decreasing sharply from 459.95 s in epoch 1 to approximately 452 s for all remaining epochs.

This difference is explained by the "pipeline fill" phenomenon. During the first forward passes, only the first stage has active microbatches, while downstream stages remain idle until enough microbatches have propagated through the model. Once the pipeline contains m in flight microbatches, all four GPUs operate concurrently, and the pipeline "drain" at the end of the epoch constitutes only a small fraction of total time. The consistent per epoch timing after epoch 2 is direct evidence that bubble overhead is reduced to a minimum once steady operation is reached.

Interestingly, this behavior was also visible in `watch -n1 nvidia-smi`. During the first ten to fifteen seconds of training, GPU 0 reached its peak memory allocation immediately (approximately 19 GB), while GPUs 1–3 reported substantially lower allocations. As the pipeline filled, their memory usage rose until all four devices stabilized at nearly identical VRAM usage. This hardware observation confirmed the measured throughput trend: the system transitions from a partially filled pipeline to a fully utilized one, precisely matching the behavior predicted by microbatch pipeline theory.

Throughput Stability Indicator. The remarkable consistency of throughput from epochs 2 to 10 (variation below 0.5% in Table 1) also indicates that the custom pipeline implementation correctly balanced stage workloads. In poorly partitioned pipelines, throughput tends to oscillate significantly as one stage consistently bottlenecks the others.

This uniformity is particularly notable given the transformer depth of ViT-H/14 and the nontrivial variation in compute cost per layer. Achieving this level of stage balance is nontrivial and strongly supports the claim that careful manual partitioning can outperform automated splitting heuristics, especially on moderately sized clusters where

a single heavy stage can single-handedly stall the entire pipeline.

ViT-L/16 with Data Parallelism

In contrast, Table 2 shows the largest configuration that can comfortably fit within single-GPU memory constraints, allowing us to leverage DP's superior efficiency.

Table 2. Training Progress: ViT-L/16 with Data Parallelism (4 GPUs)

Epoch	Loss	Accuracy	Throughput	Time
1	1.9716	28.68%	125.71	397.09s
2	1.5833	42.86%	125.60	397.47s
3	1.4236	48.52%	125.49	397.79s
4	1.2936	53.10%	125.48	397.83s
5	1.2219	55.85%	125.51	397.73s
6	1.1382	58.81%	125.51	397.75s
7	1.0672	61.76%	125.60	397.44s
8	1.0175	63.49%	125.65	397.28s
9	0.9637	65.50%	125.78	396.89s
10	0.9204	67.08%	125.87	396.61s

Cumulative time: 3973.88s. The DP training demonstrates several advantages: higher throughput (125-126 vs 108-110 samples/second), faster epoch times (397s vs 452s), and significantly better convergence 67.08% accuracy compared to 13.27% for ViT-H/14 (which is 2 times larger model). The training data shows rapid improvement in early epochs, with accuracy increasing by over 14 percentage points between epochs 1 and 2, indicating efficient gradient synchronization across GPUs.

Interpreting the DP Throughput. Table 2 shows that DDP achieves not only higher throughput but also extremely stable performance across epochs. The throughput range of 125.48–125.87 img/s is narrower than that of PP, reflecting the fact that DP does not suffer from pipeline warm up. Because each GPU executes full forward and backward passes independently before gradient synchronization, all devices begin computing immediately at the start of each epoch.

The convergence behavior is also dramatically different: accuracy rises from 28.68% in epoch 1 to 67.08% in epoch 10. This 38.4 percentage point improvement stands as a strong contrast to the slow progress of the ViT-H/14 pipeline run. Crucially, this should not be interpreted as a failure of PP itself but rather as a consequence of model size. The ViT-L/16 model has roughly half the parameters of ViT-H/14 and is therefore significantly easier to optimize on CIFAR-10, especially with effective batch sizes of 192.

Furthermore, the time per epoch values (396.6–397.8 s) are lower and more stable than those in the PP case. The small fluctuations show that the compute to communication ratio remains consistent and well within the comfortable regime for DP on four A5000 GPUs. These stable numbers also indirectly confirm the absence of gradient scaling issues: if AMP or the optimizer were experiencing underflows/overflows, timing irregularities would often appear due to stalls, retries, or abnormal CUDA kernel behavior.

5. Discussion

The results across ResNets and Vision Transformers suggest that no single parallelization strategy is universally optimal. Instead, the right choice depends on a combination of model size, input resolution, and hardware constraints. For relatively small models that already fit comfortably on a single GPU, such as ResNet variants on CIFAR-10, pipeline parallelism is often unnecessary and can even be harmful. The additional complexity of managing stages and microbatches introduces overhead that is not compensated by gains in utilization.

For very large models like ViT-H/14, data parallelism alone runs into a hard memory ceiling. Even with AMP and careful memory budgeting, the combination of parameter, optimizer, and activation states exceeds what a single GPU can hold at reasonable batch sizes. Pipeline parallelism then becomes less of a performance optimization and more of a prerequisite: without it, the model simply cannot be trained. The ViT-L/16 vs. ViT-H/14 comparison concretely illustrates this trade off: DP delivers higher throughput for the smaller model, but PP is the only way to fit the larger model on the same hardware.

Another recurring theme is that the theoretical models for utilization and memory usage are useful but incomplete. Simple formulas such as $\frac{m}{m+n-1}$ provide intuition about pipeline bubbles, and linear models for memory footprint highlight how batch size interacts with activations. However, the real system also includes subtle effects from kernel fusion, CUDA stream scheduling, communication overlap, and library level fragmentation.

To conclude, when a model cannot be trained at all with DP due to memory constraints, throughput comparisons become irrelevant. PP's ability to train ViT-H/14, and achieving consistent loss reduction and accuracy improvement, demonstrates its value as a memory scalability solution, even at the cost of reduced throughput.

6. Computational Resources

The project uses shared access to the University of Rochester GPU cluster. All experiments run on a multi-GPU node with:

- $4\times$ NVIDIA RTX A5000 (24 GB),
- Code and configuration available here

References

[1] Tianqi et al. Chen. Training deep nets with sublinear memory cost. *arXiv:1604.06174*, 2016.

[2] Aaron et al. Harlap. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

[3] Yanping Huang, Youlong Cheng, and Ankur et al. Bapna. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *NeurIPS*, 2019.

[4] Shen et al. Li. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv:2006.15704*, 2020.

[5] Deepak et al. Narayanan. Efficient large-scale language model training on gpu clusters. *USENIX OSDI*, 2021.

[6] Samyam et al. Rajbhandari. Zero: Memory optimization towards training a trillion parameter models. *SC*, 2020.