Scalable Machine Learning: Distributed Training Methods for Large Models and Datasets

Presentation by:

- Mohamed STIFI

10/01/2025

Overview

01

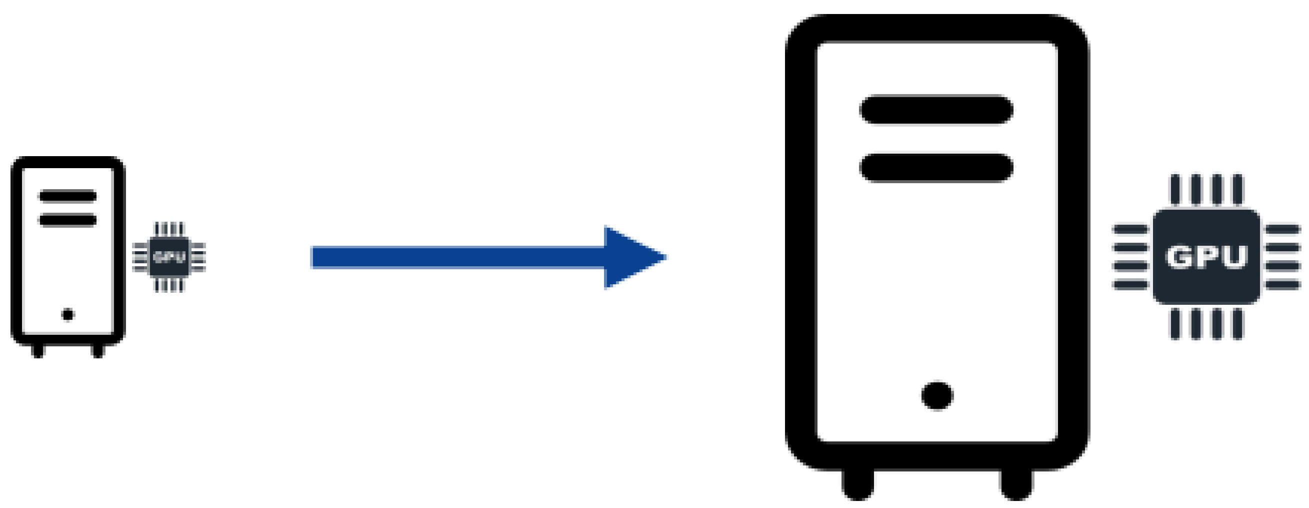
distributed training

01

What is distributed training

Vertical scaling

No code change

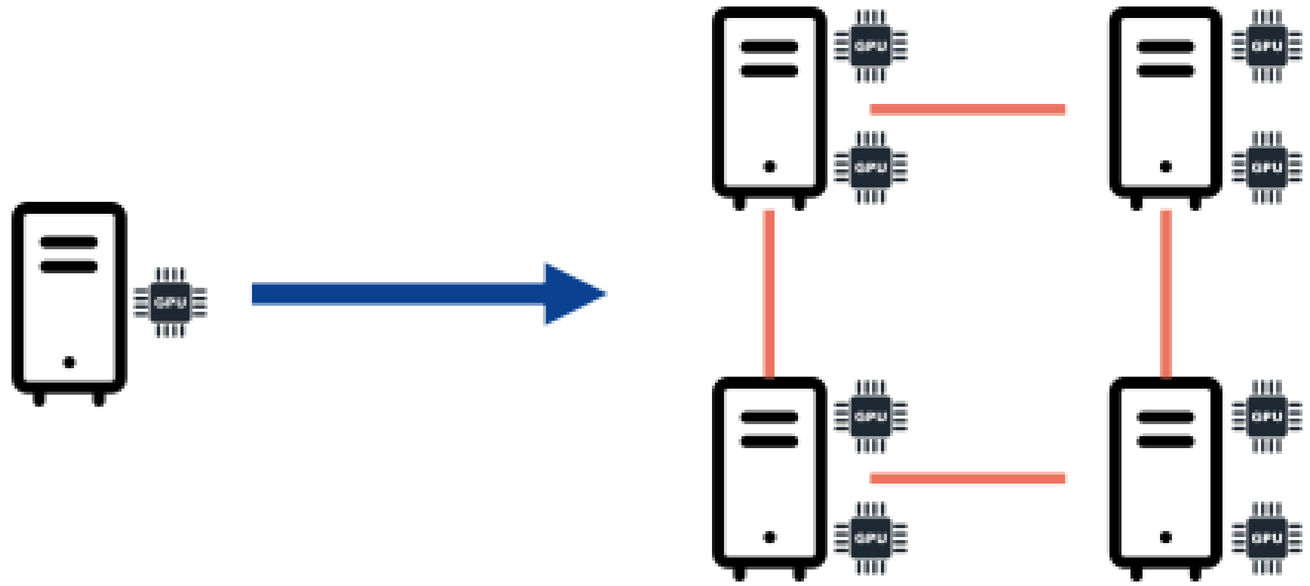


1x Server
8GB RAM
4GB GPU Memory

1x Server
64GB RAM
32GB GPU Memory

Horizontal scaling

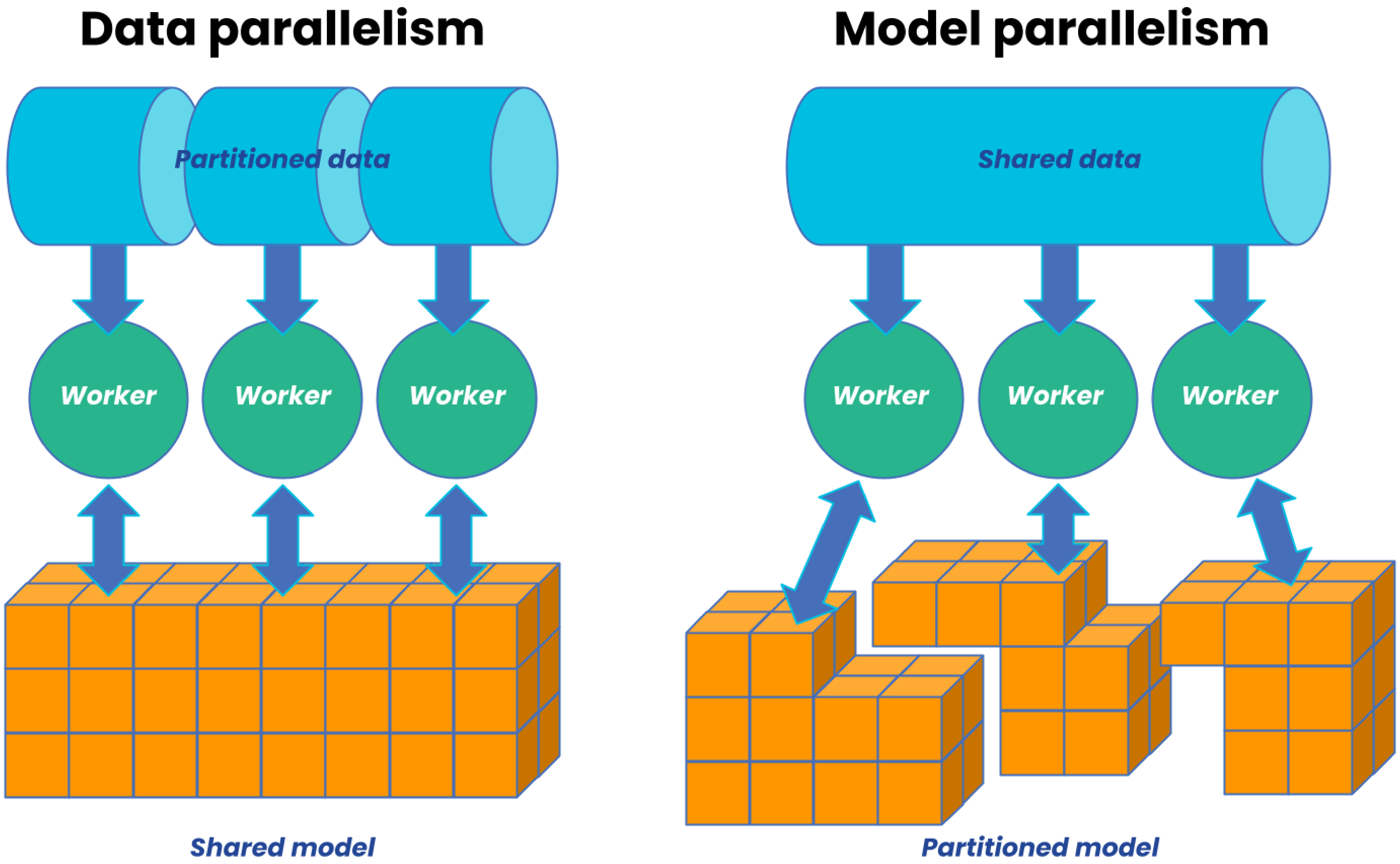
Minimal code change (thanks to PyTorch)



1x Server
8GB RAM
4GB GPU Memory

4x Servers
8GB RAM
4GB GPU Memory (x2)

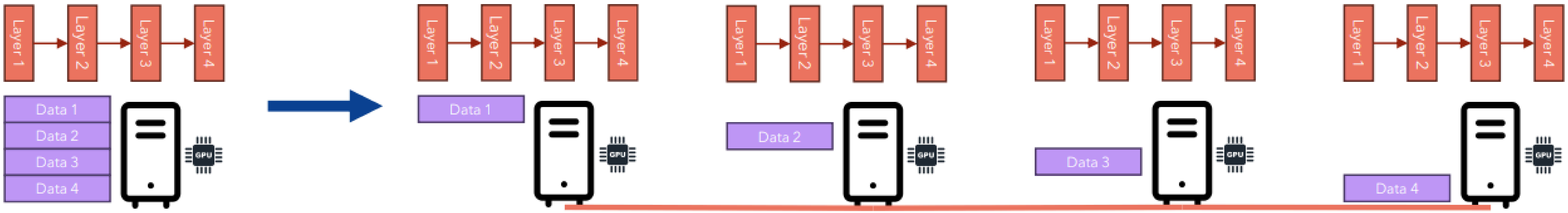
01 Data Parallelism vs Model Parallelism



01

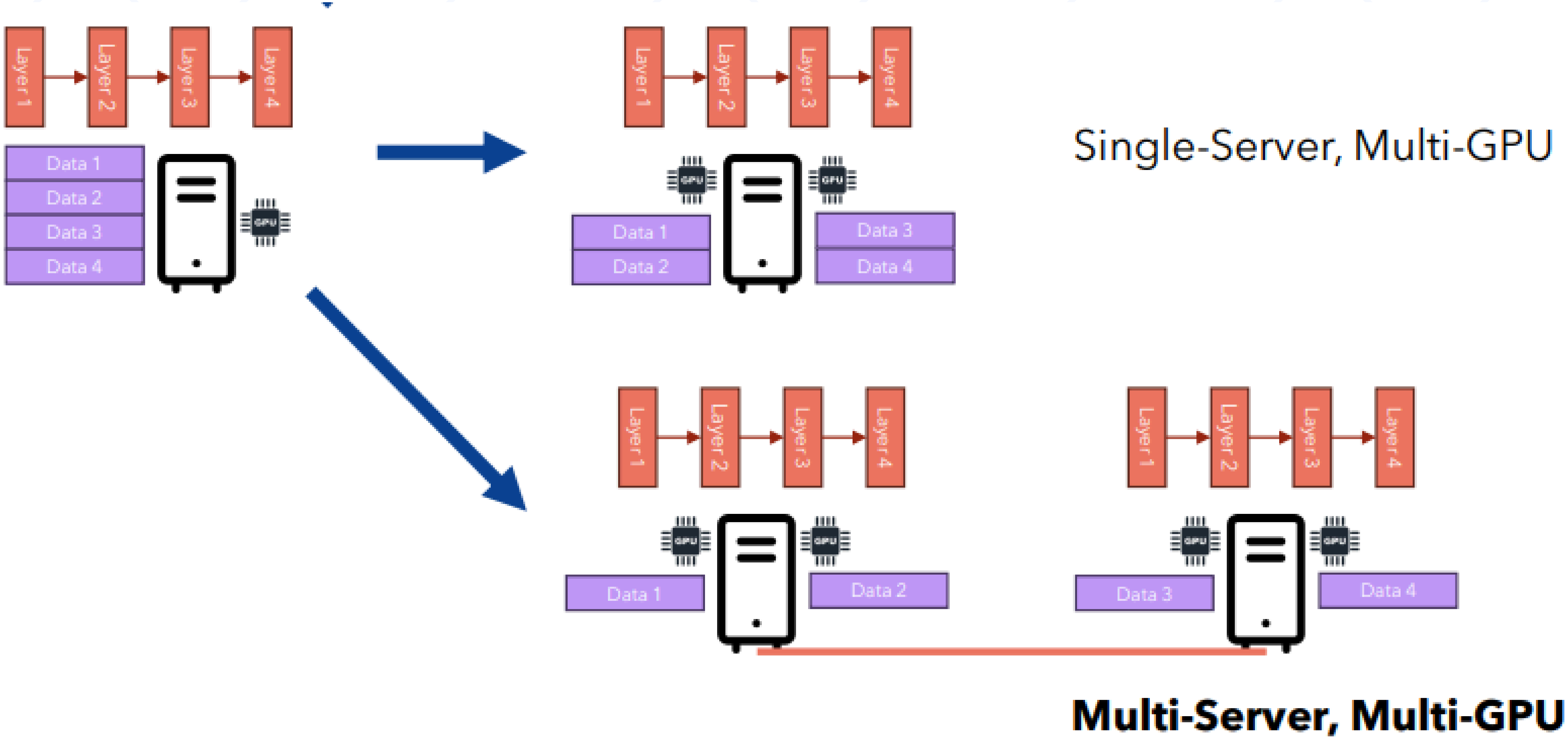
Data Parallelism

If the model can fit within a single GPU, then we can distribute the training on multiple servers (each containing one or multiple GPUs), with each GPU processing a subset of the entire dataset in parallel and synchronizing the gradients during backpropagation. This option is known as Data Parallelism.



01

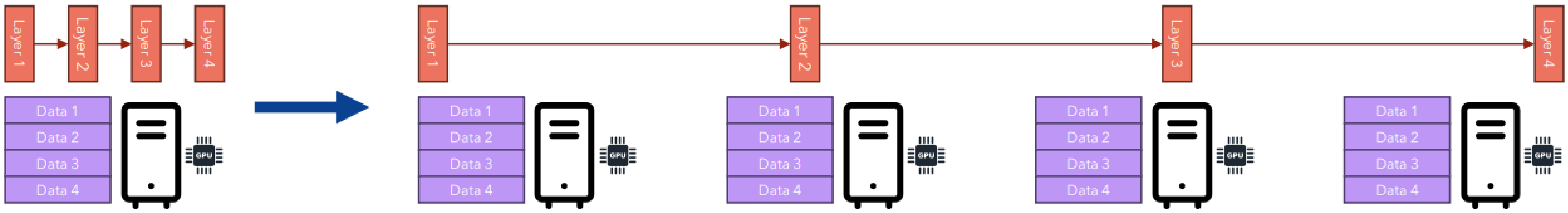
Types of Data Parallelism



01

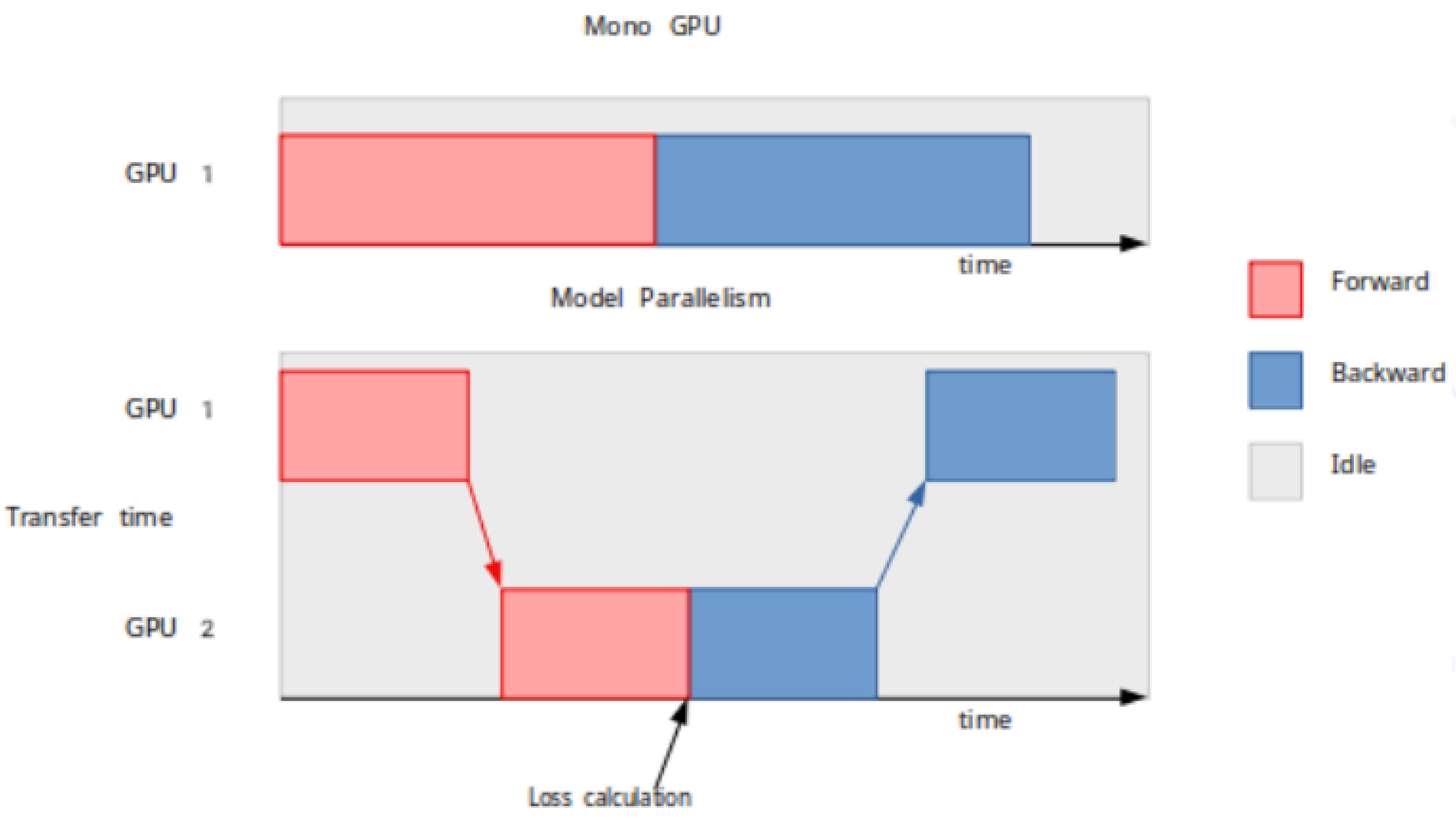
Model Parallelism

If the model cannot fit within a single GPU, then we need to “break” the model into smaller layers and let each GPU process a part of the forward/backward step during gradient descent. This option is known as Model Parallelism.



01

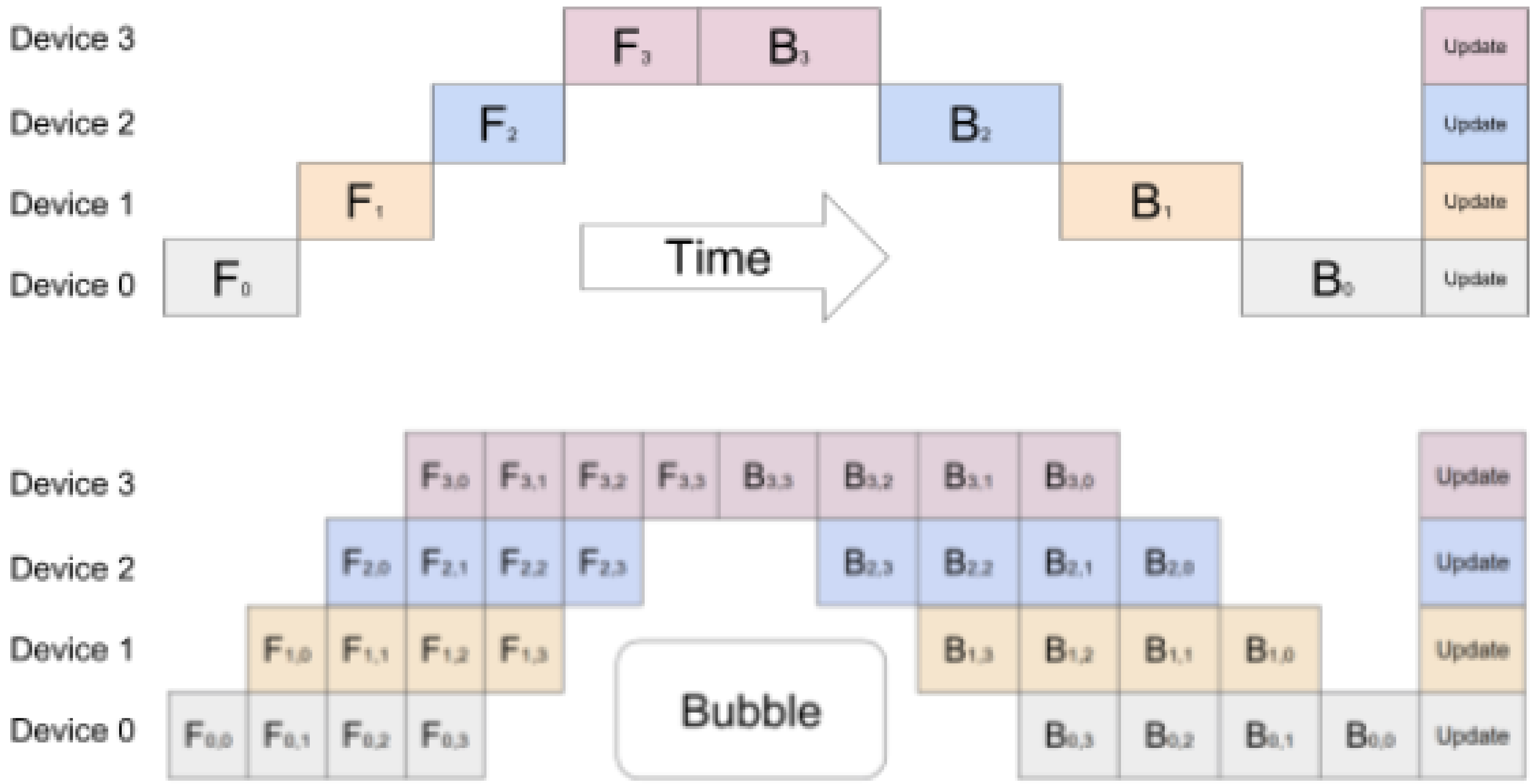
Types of Model Parallelism



Layer-wise Parallelism

01

Types of Model Parallelism



Pipeline Parallelism

01 Algorithm

Algorithm 1 Algorithm for Distributed Training in Machine Learning

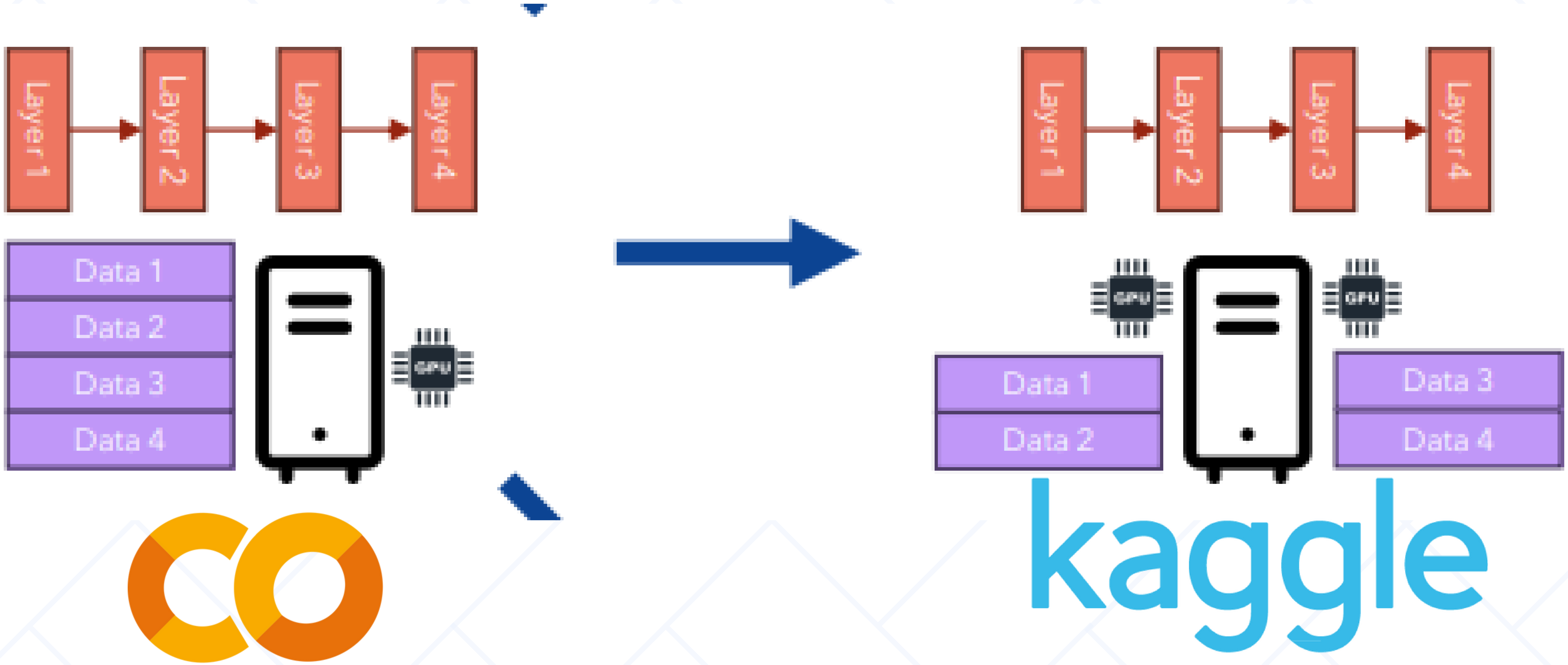
Require: Number of GPUs $world\_size$, Dataset D, Number of Epochs num_epochs, Learning Rate LR
Ensure: Trained Model M

Set Up the Environment : ▷ Import libraries and define global variables
Initialize the Distributed Data Parallel (DDP) Environment :
  Define and execute functions :
    `setup(rank, world_size)`: Initialize process group
    `cleanup()`: Destroy process group
Define a Model :
  Implement architecture with `Model(nn.Module)`
  For model parallelism, manually assign layers to devices
  `create_model()`: Return an instance of the model
Create a Dataloader :
  Partition dataset D across GPUs using `DistributedSampler`
  `create_dataloader(rank, world_size)`:
    Partition D, create mini-batches, and return dataloader instances
Implement the Training Loop :
  Define helper functions :
    `train(model, iterator, optimizer, criterion, rank)`: Single training step
    `evaluate(model, iterator, criterion, rank)`: Single evaluation step
  Define main training function `main_train(rank, world_size)`:
    a. Setup distributed process groups with `setup(rank, world_size)`
    b. Create model and dataloaders
    c. Wrap model with `DistributedDataParallel (DDP)`
    d. Define loss criterion and optimizer
    e. Train for num_epochs, compute metrics
    f. Evaluate on test set after training
    g. Cleanup environment with `cleanup()`
Main Execution :
  Define execution function :
    Set $world\_size$ (number of GPUs)
    Use `multiprocessing.spawn()` to start distributed processes
  Ensure algorithm supports model and data parallelism

02 Experimental Environment

02

Experimental Environment



02

Experimental Environment

Models:

- *VGG11*
- *MLP*
- *NBoW*
- *CNN*
- *Seq2Seq*
- *ConvAutoEncoder*

metrics:

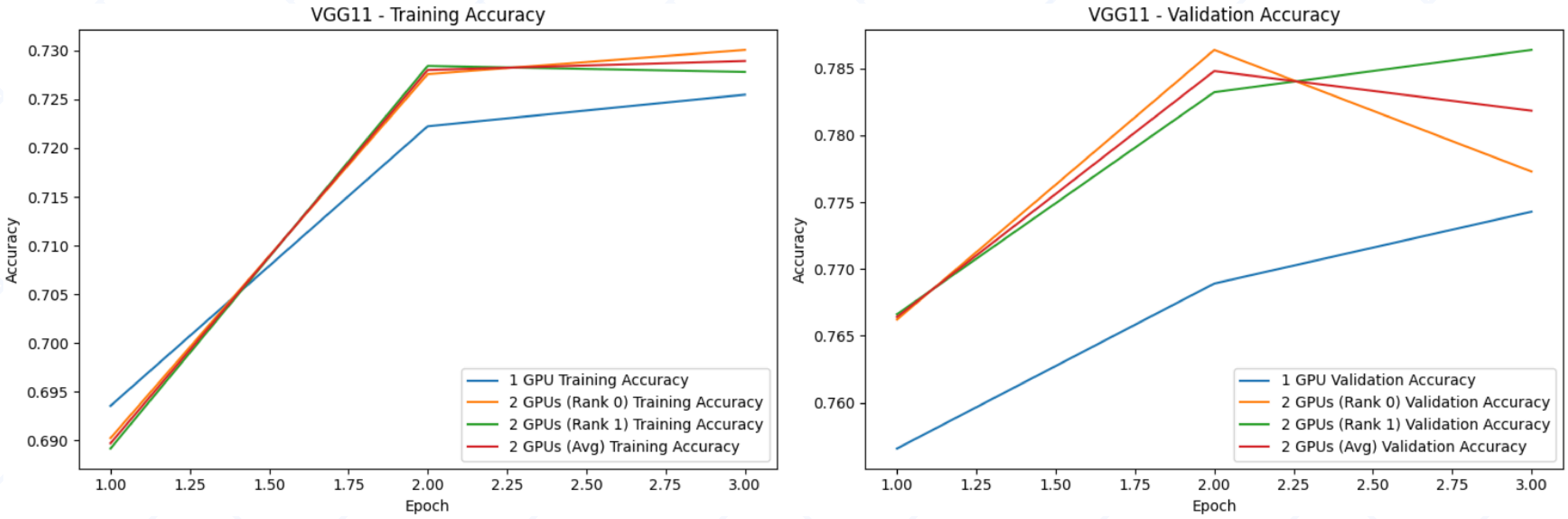
- *Training Time*
- *Evaluation Time*
- *Loss*
- *Accuracy*

03 Experimental Results

03

VGG11

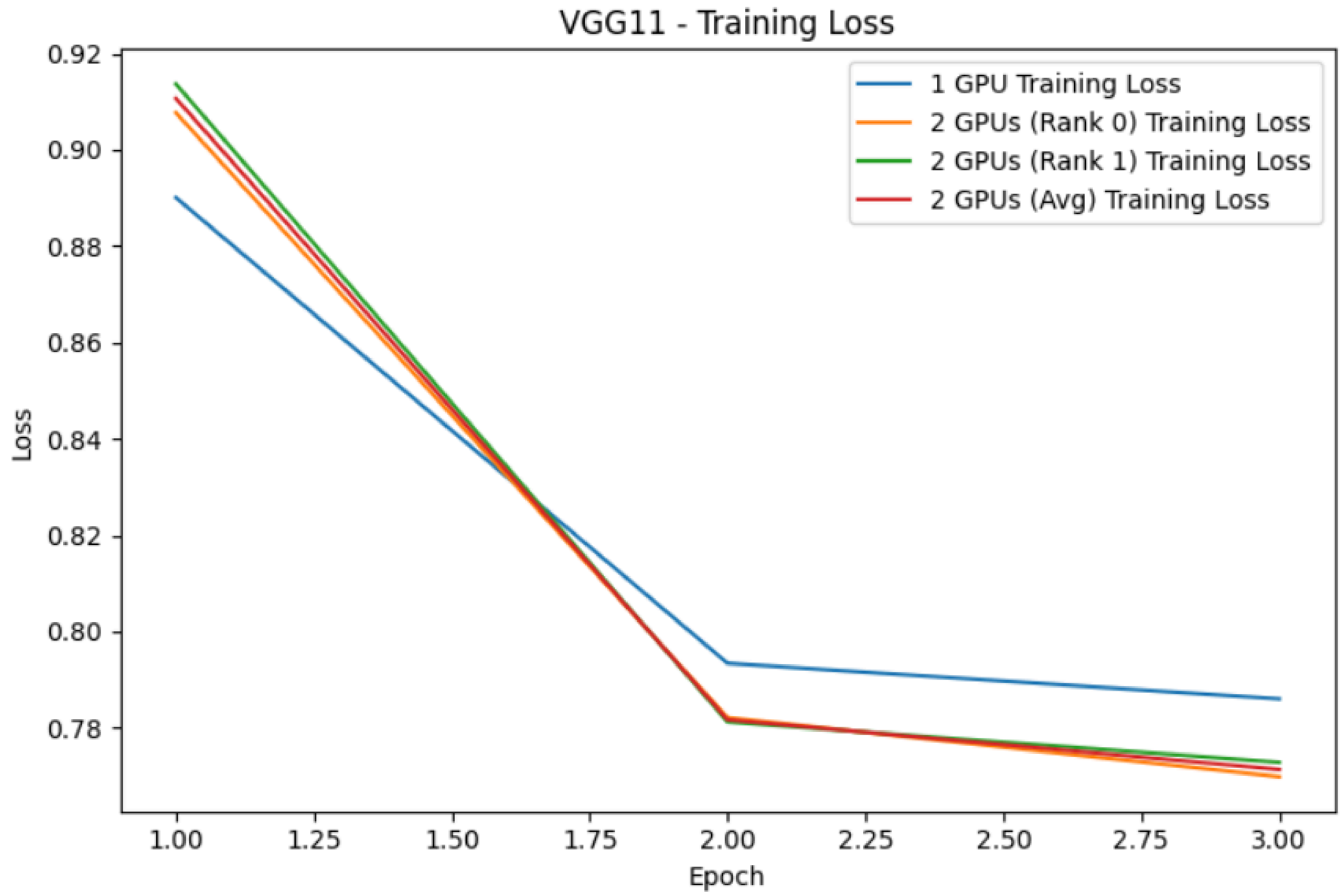
Training and Validation Accuracy

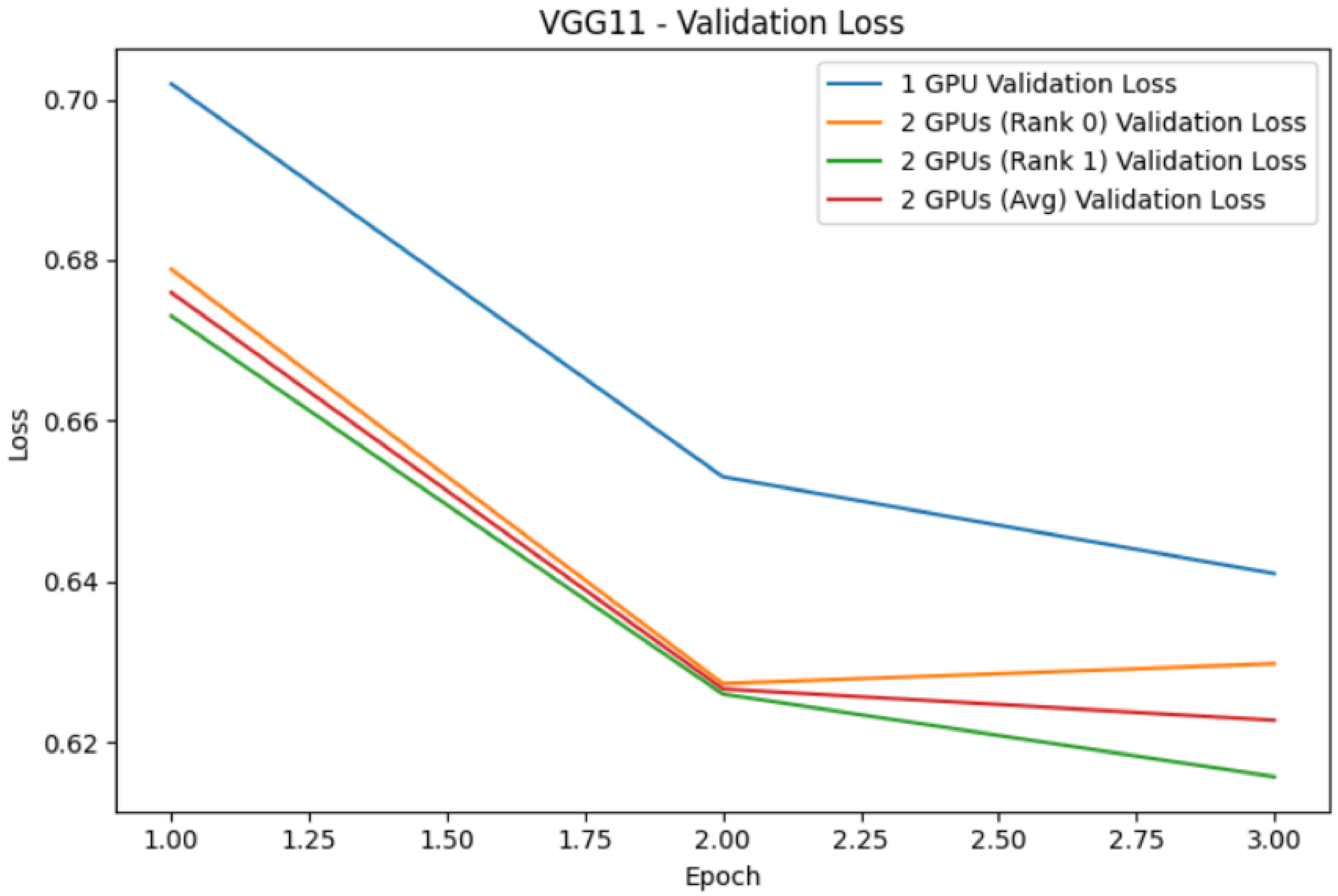


03

VGG11

Training and Validation Loss

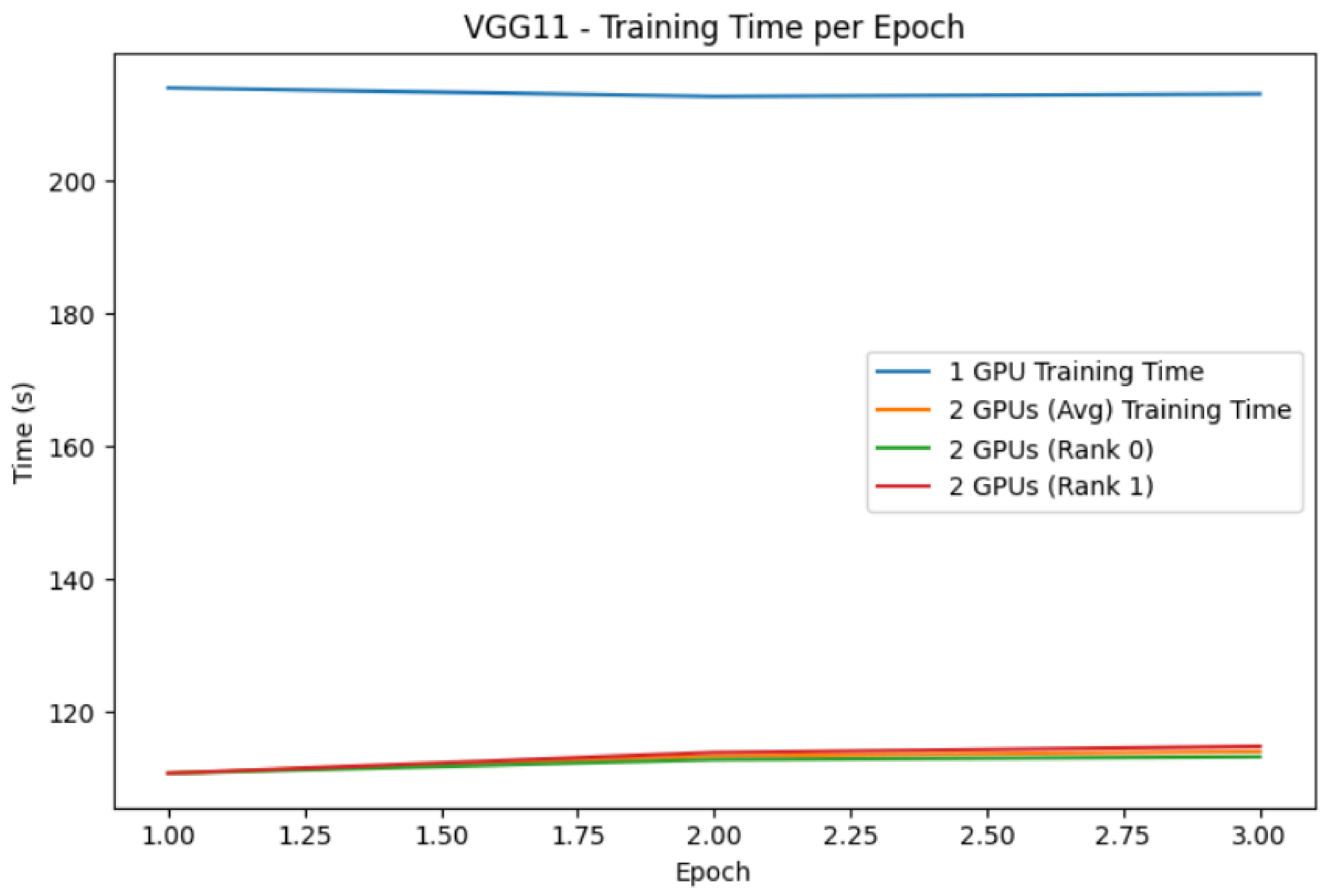


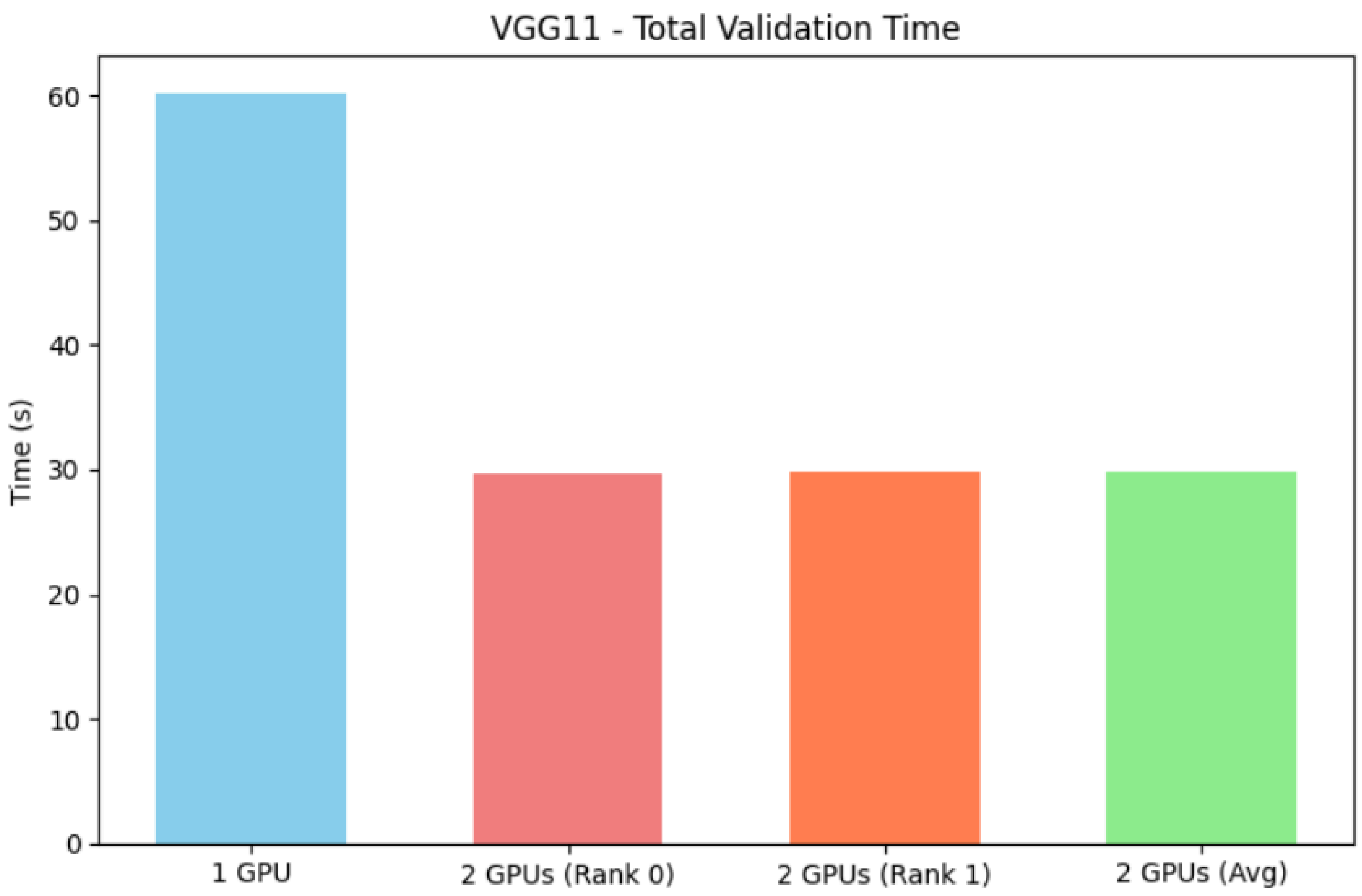


03

VGG11

Training and Validation Time

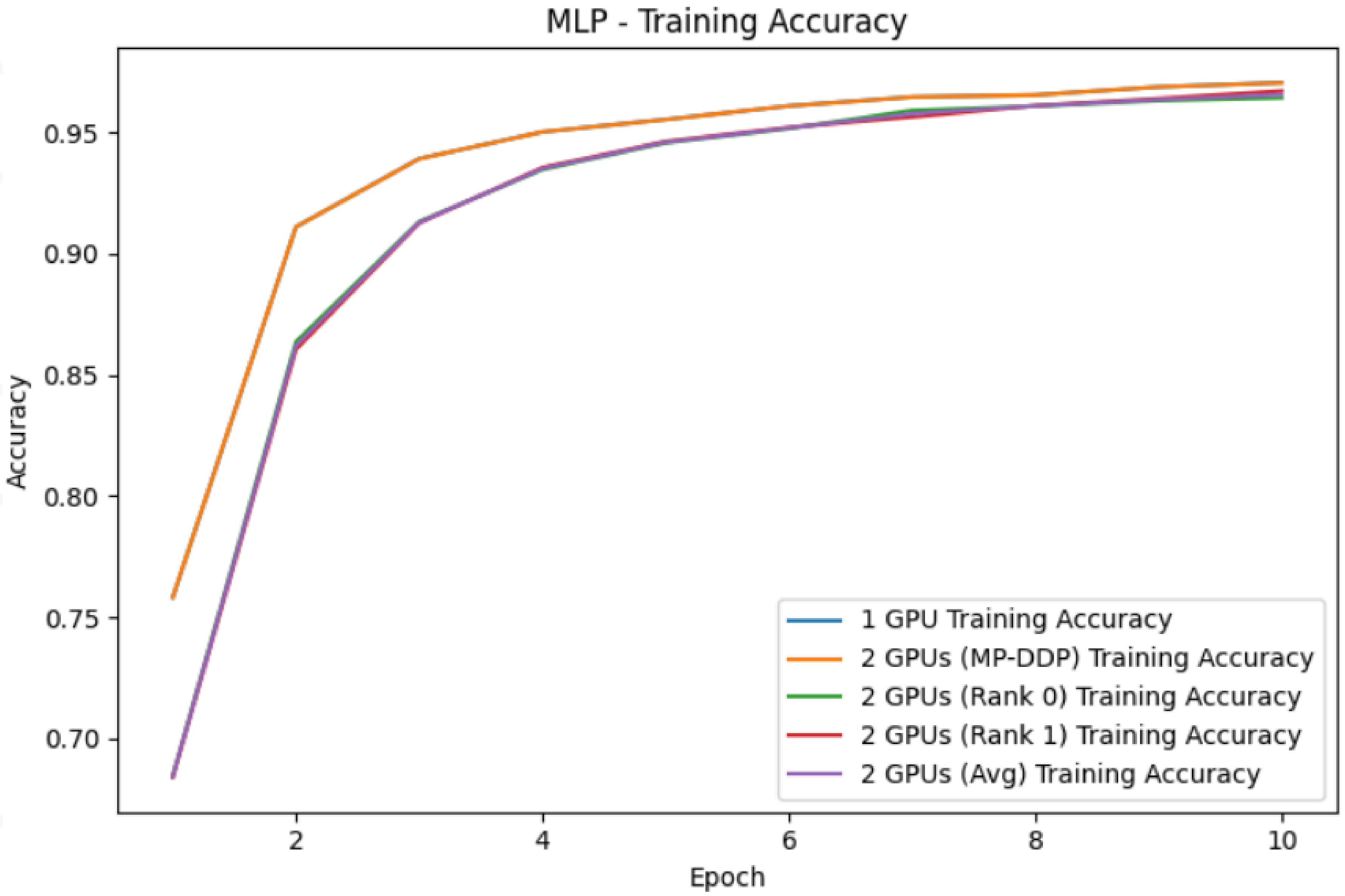


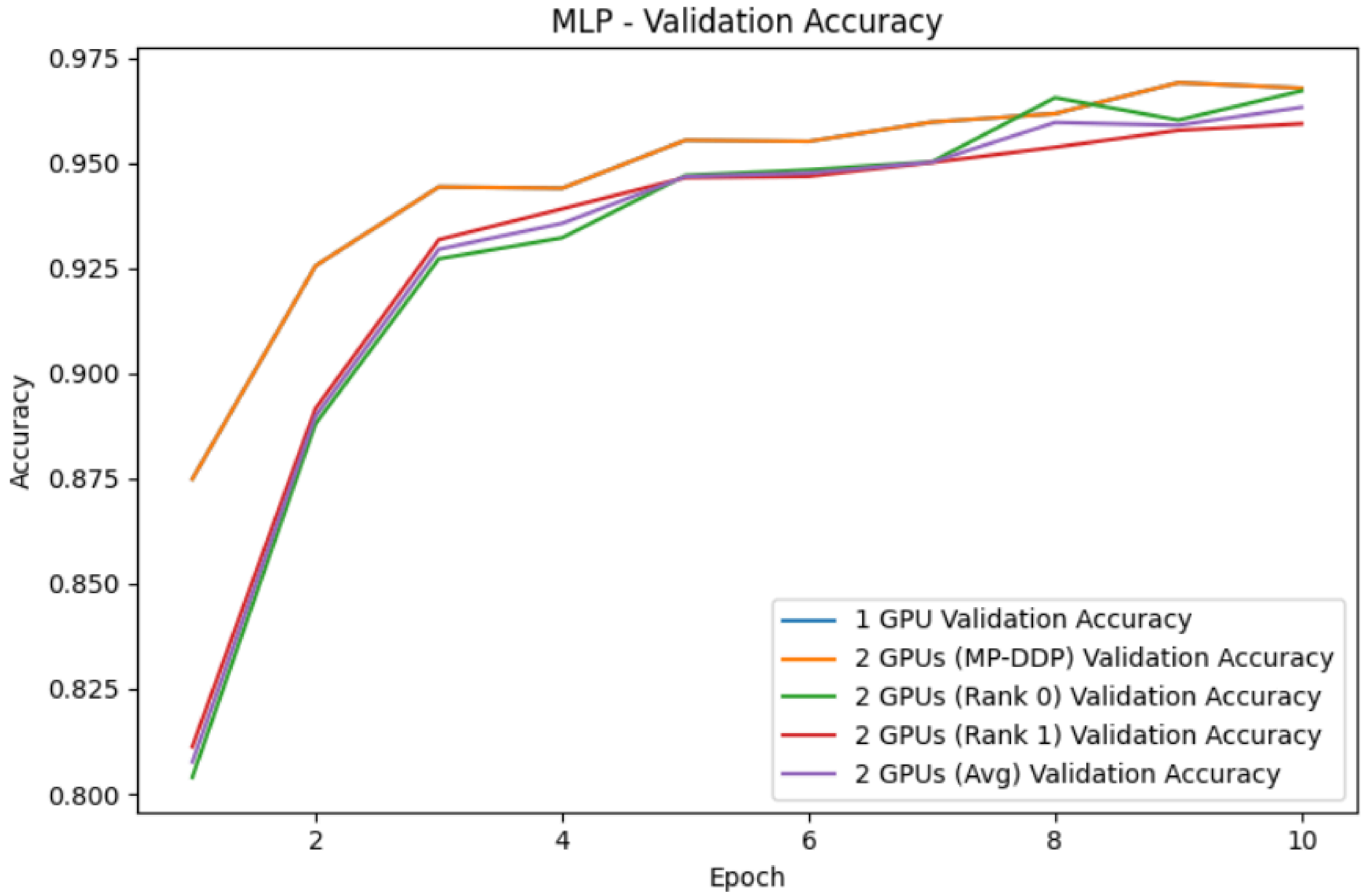


03

MLP

Training and Validation Accuracy

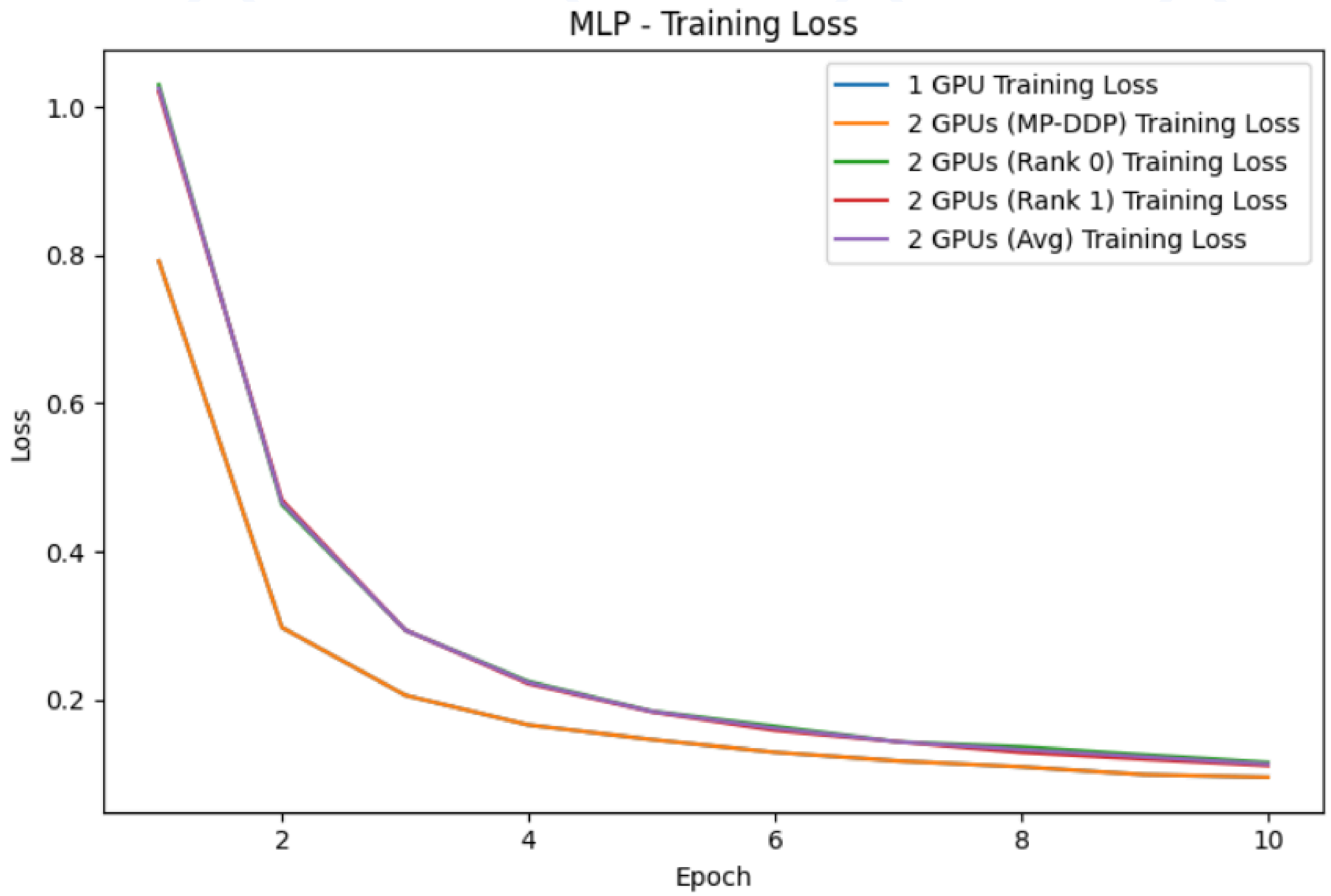


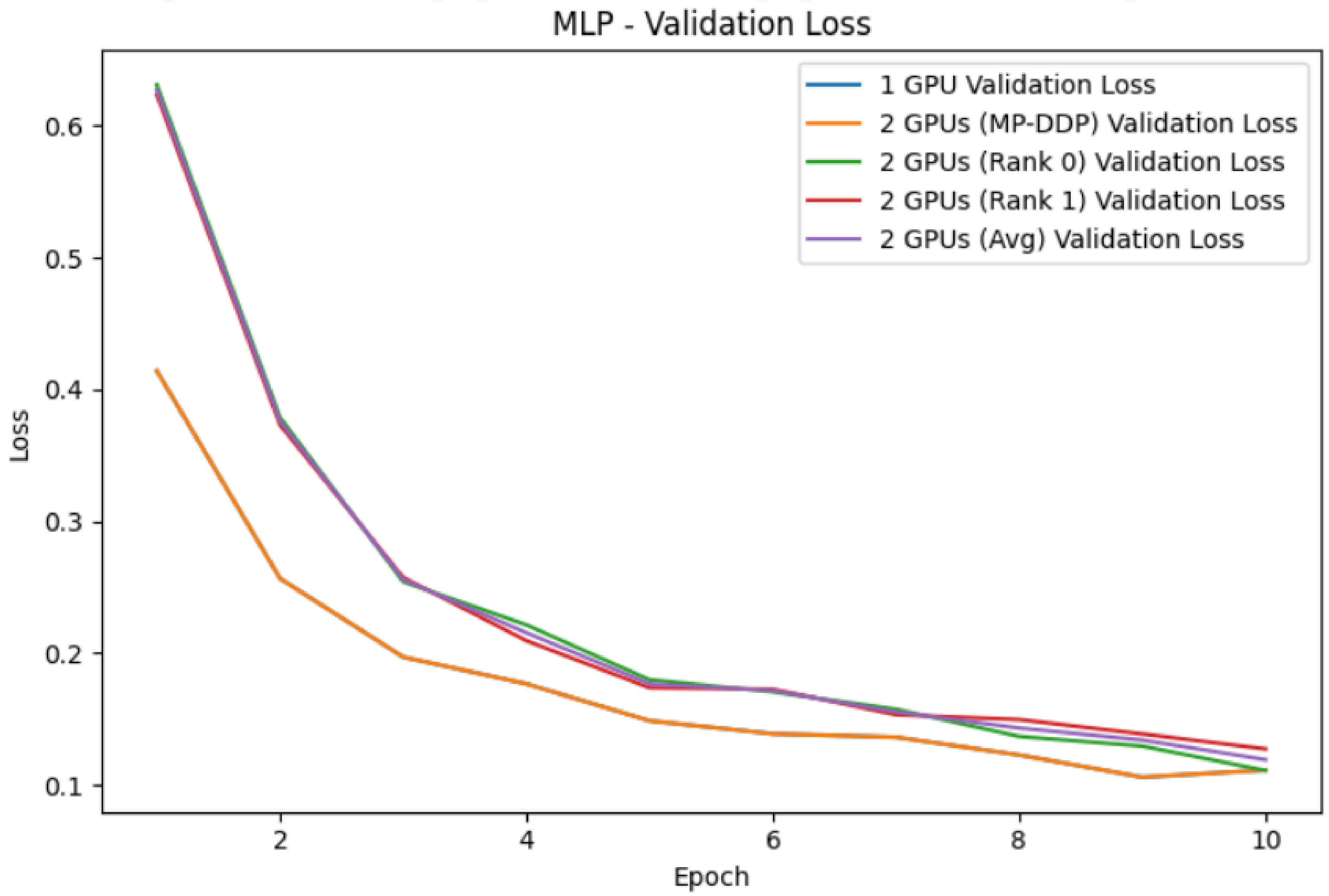


03

MLP

Training and Validation Loss

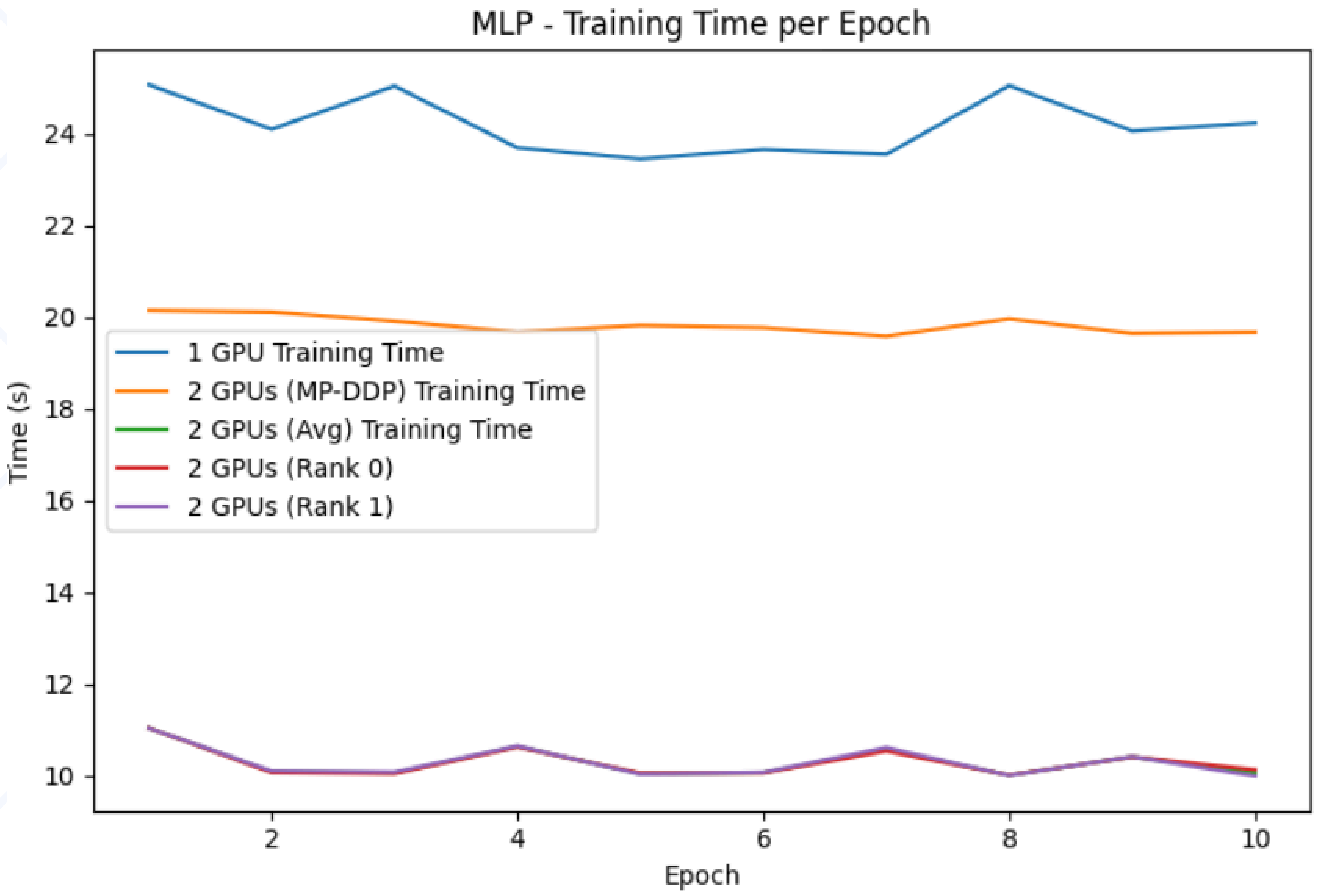


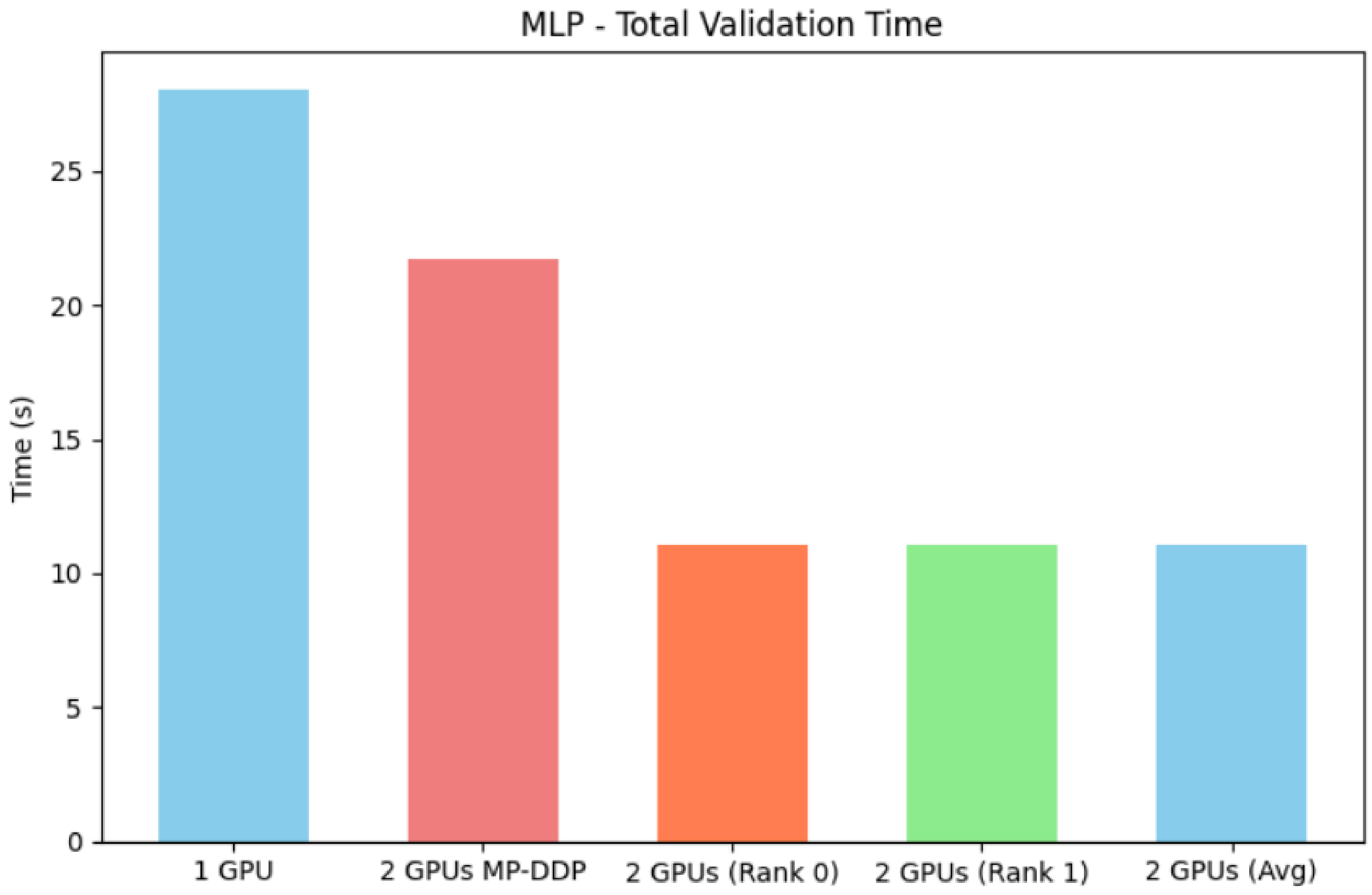


03

MLP

Training and Validation Time

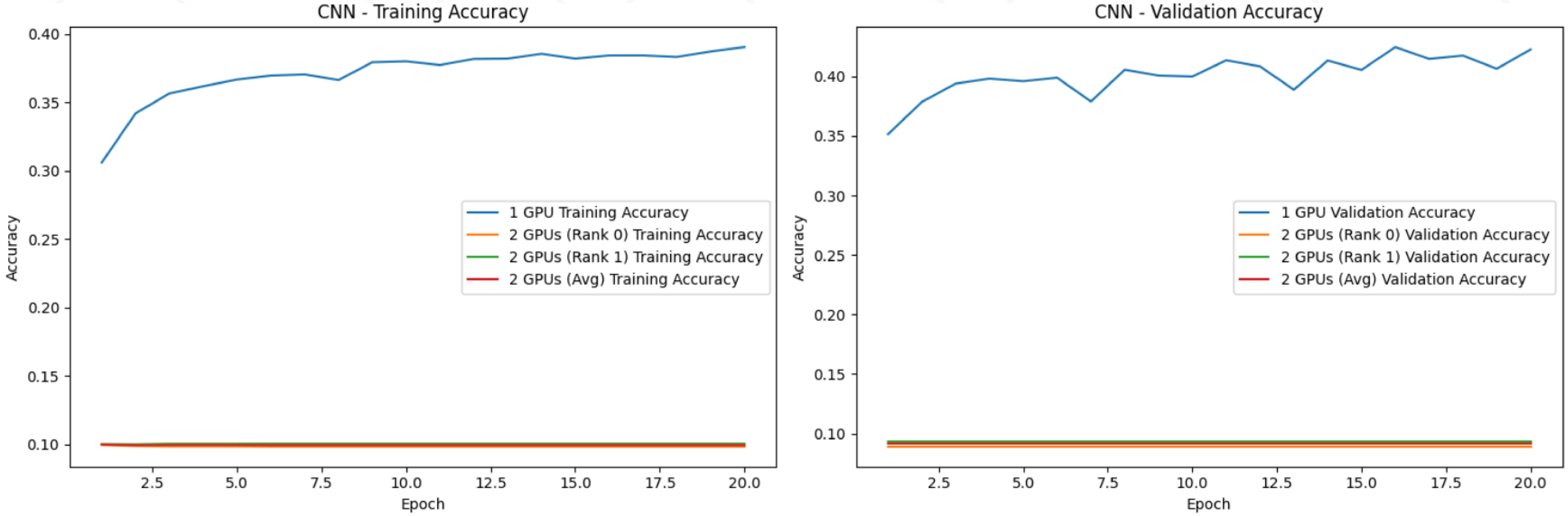




03

CNN

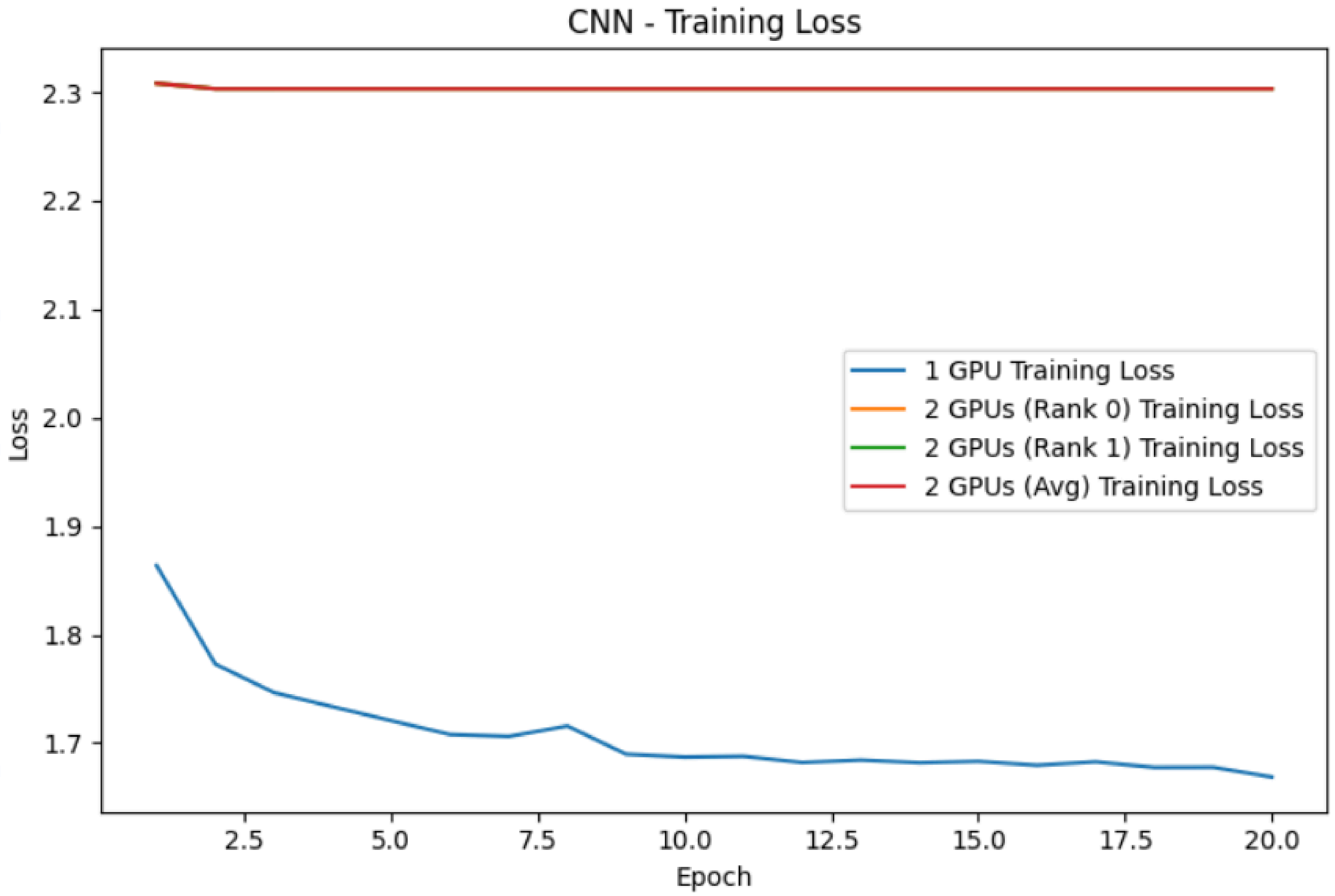
Training and Validation Accuracy

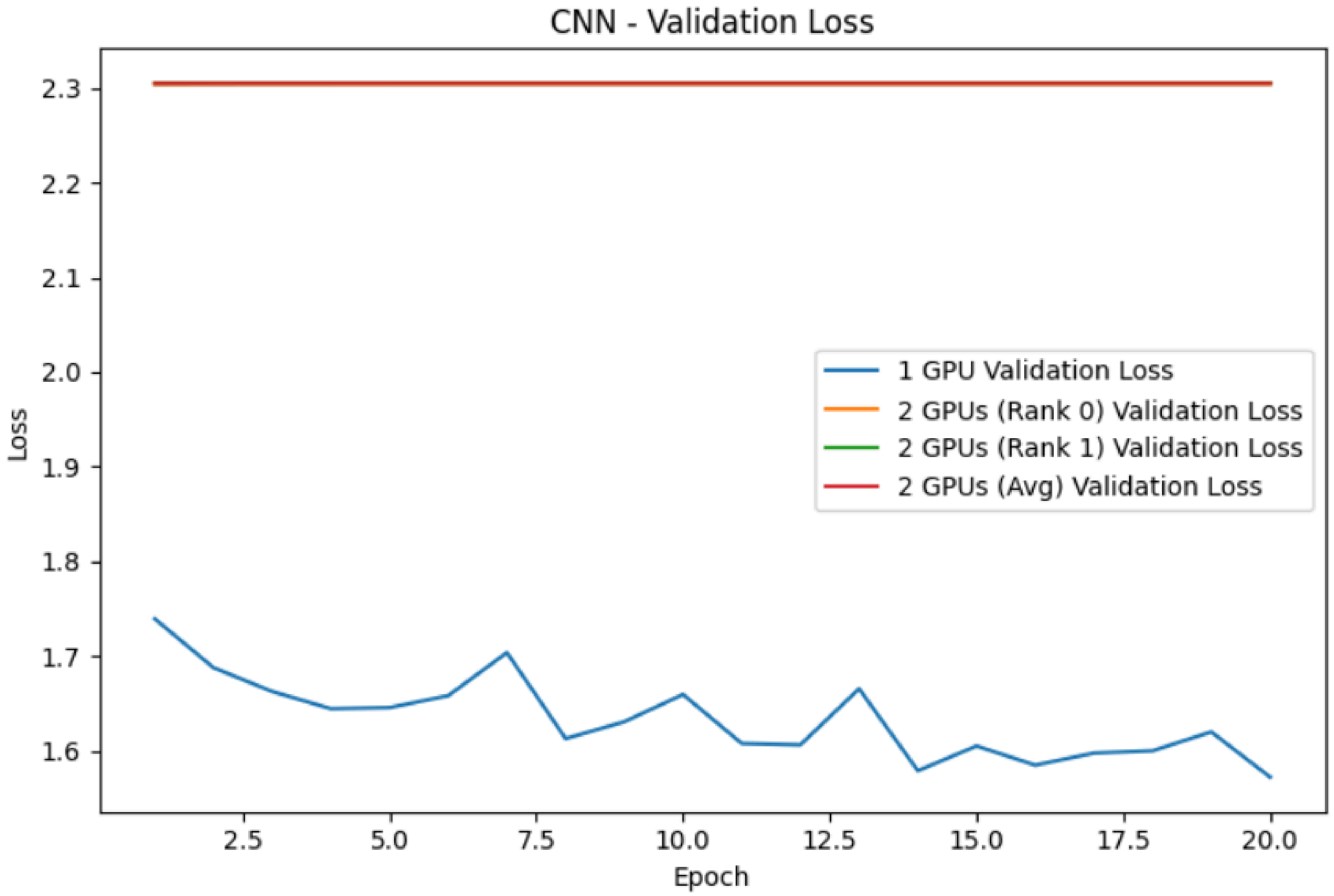


03

CNN

Training and Validation Loss

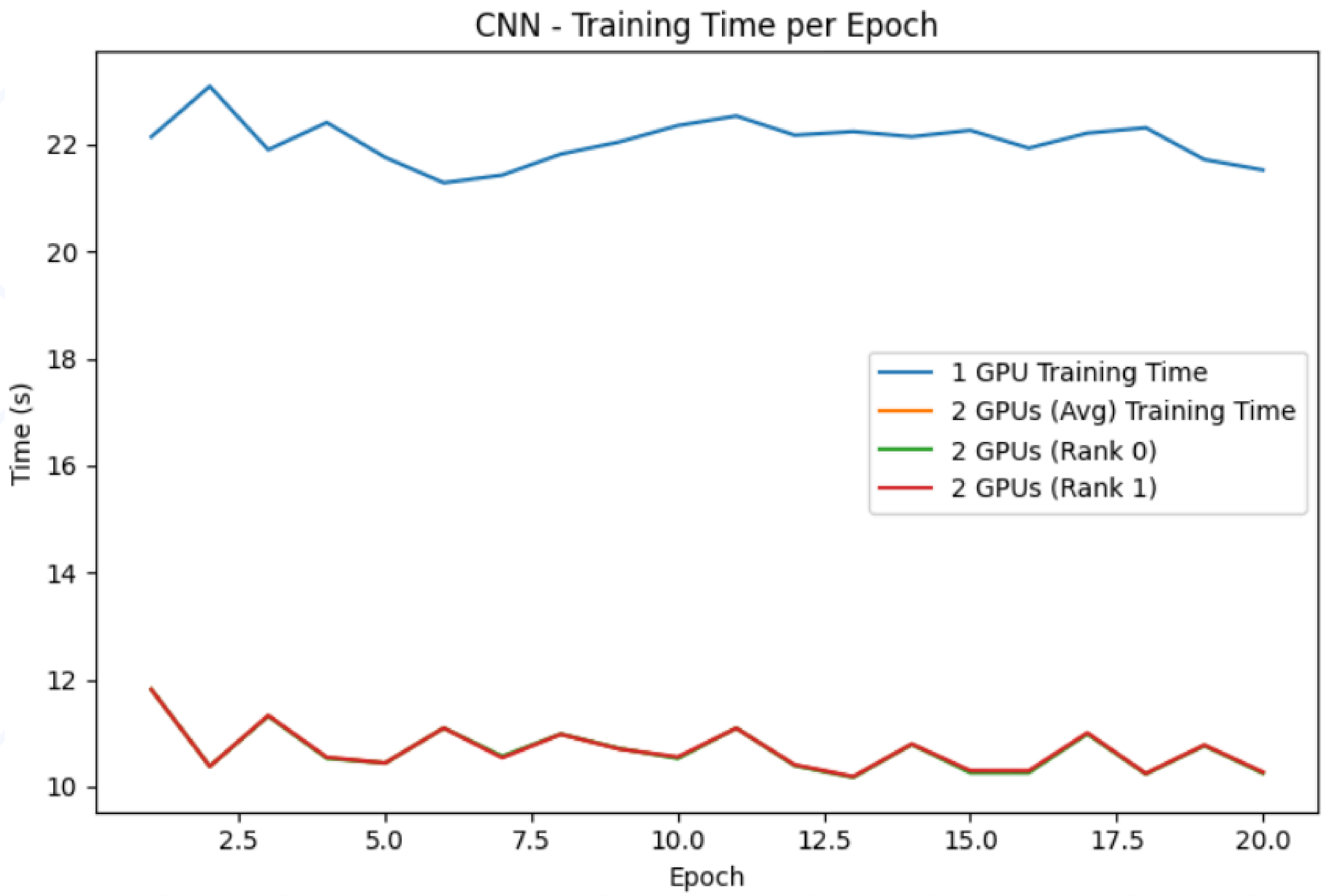


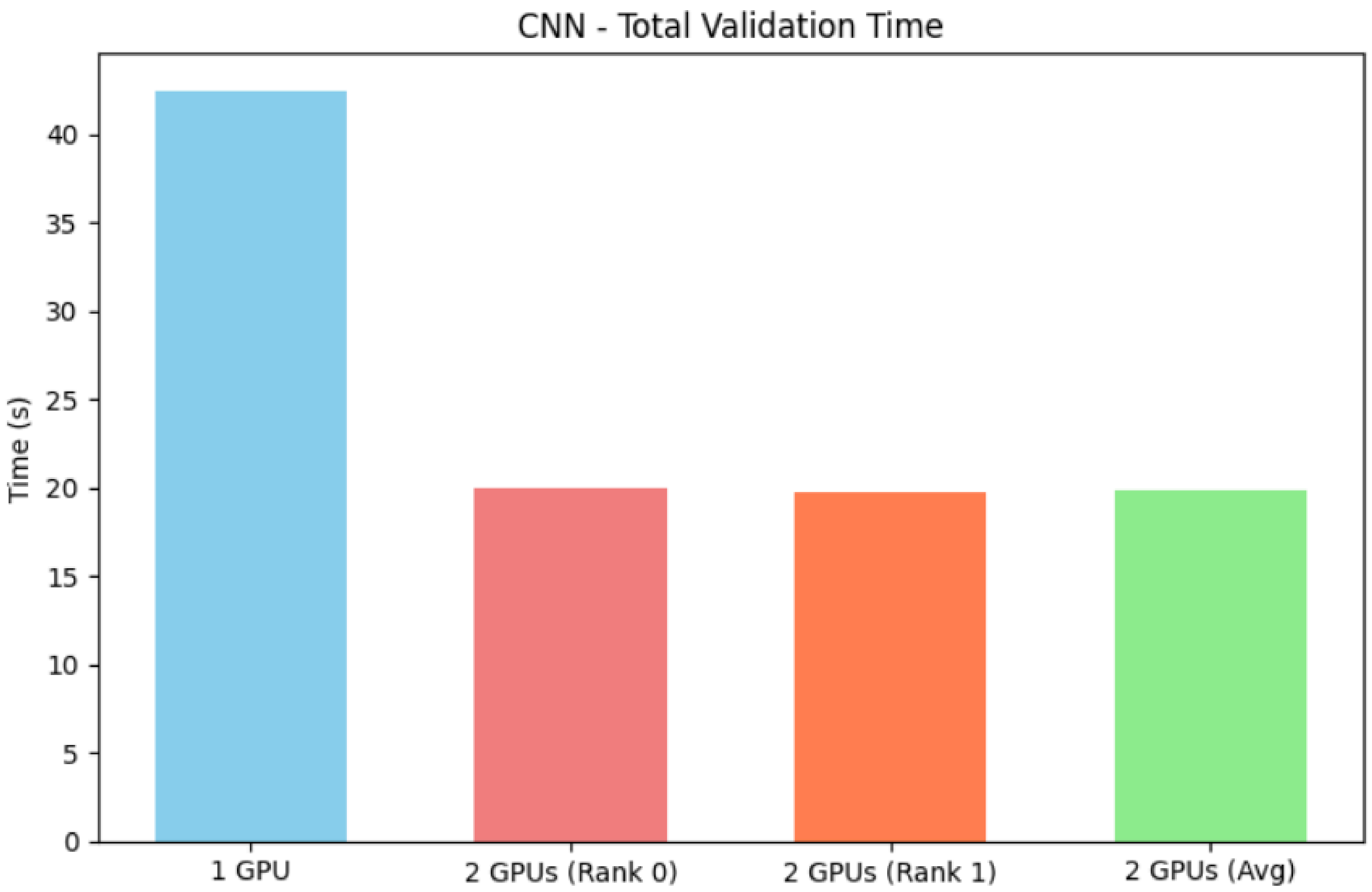


03

CNN

Training and Validation Time

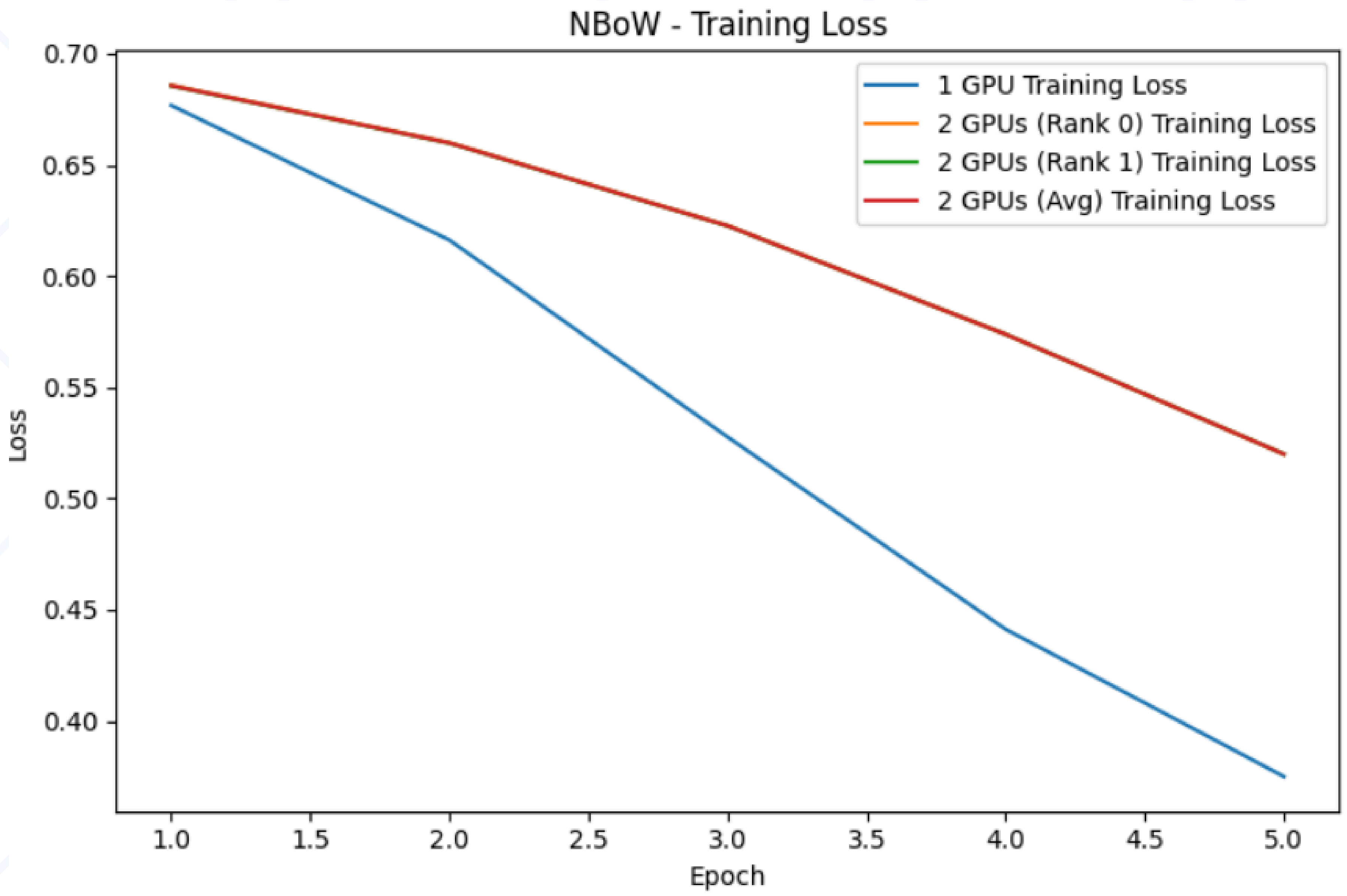


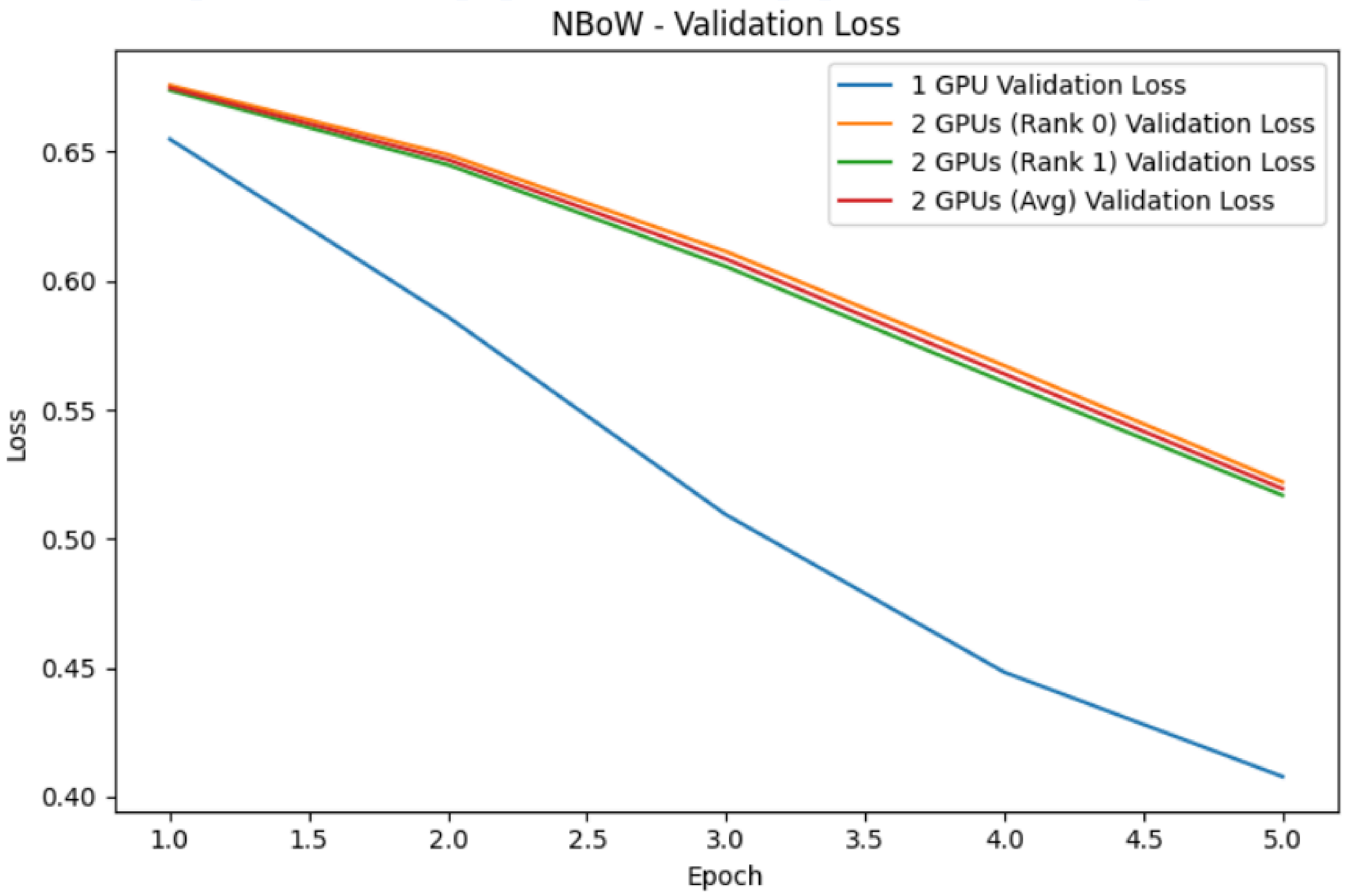


03

NBoW

Training and Validation Loss

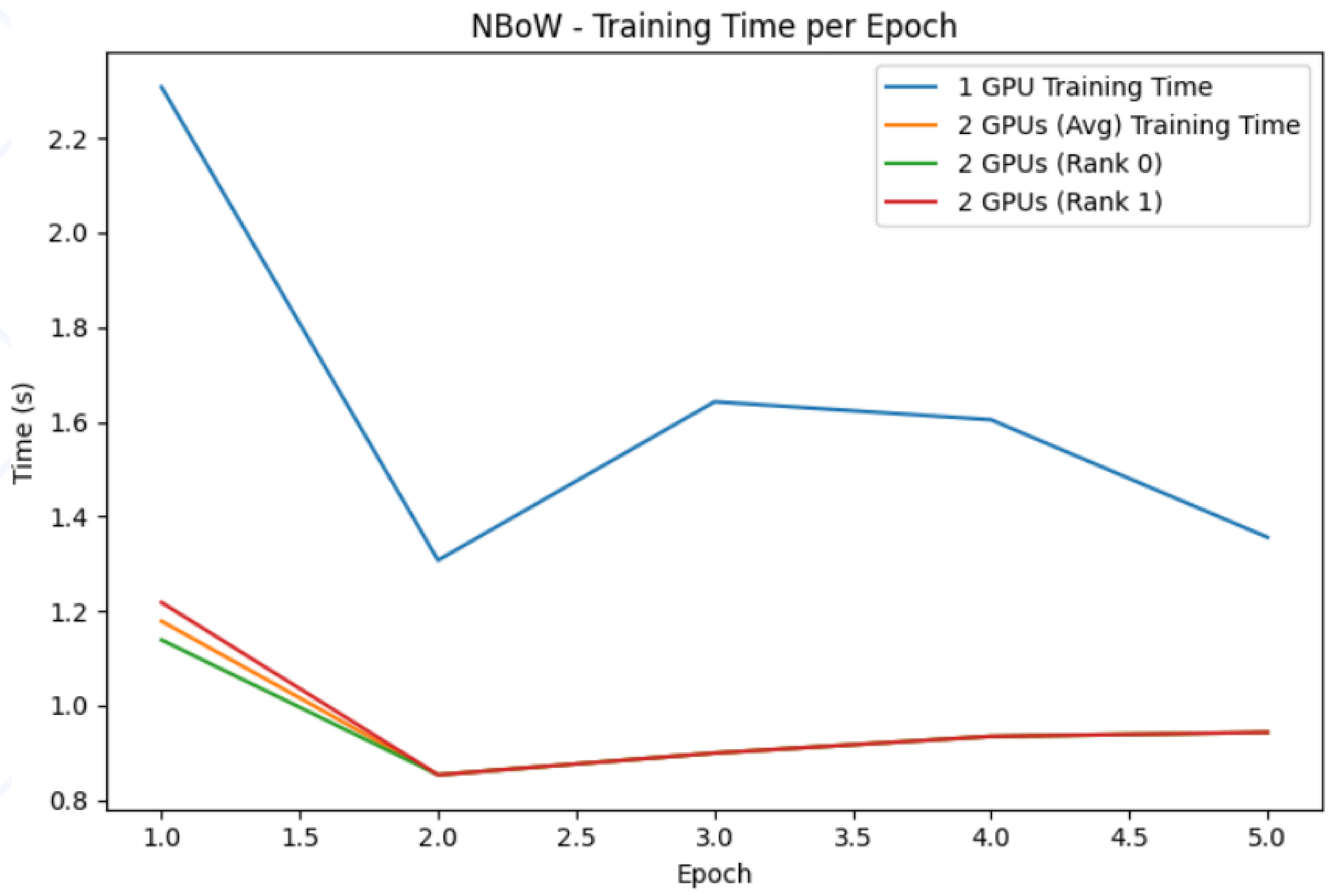


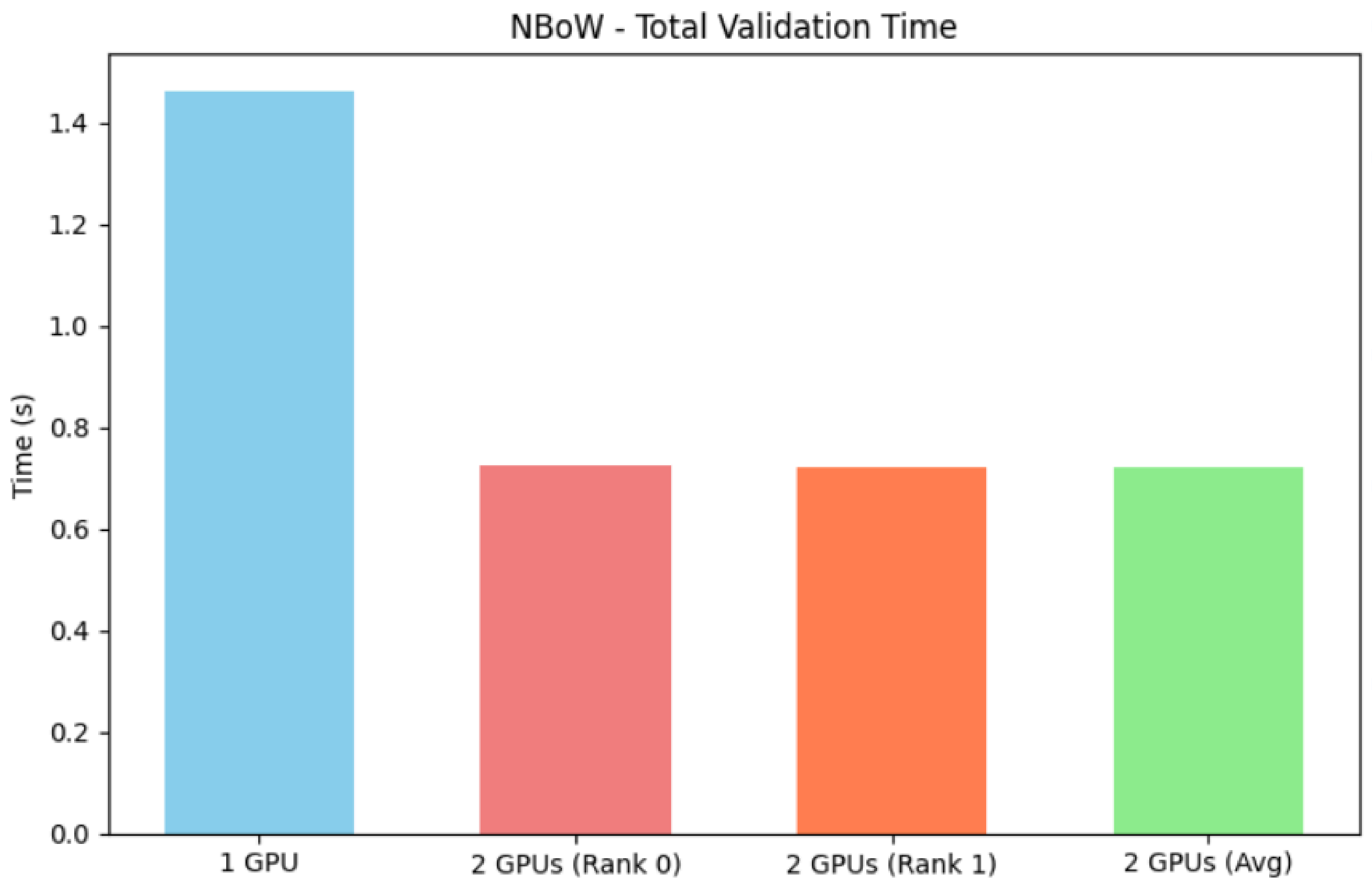


03

NBoW

Training and Validation Time

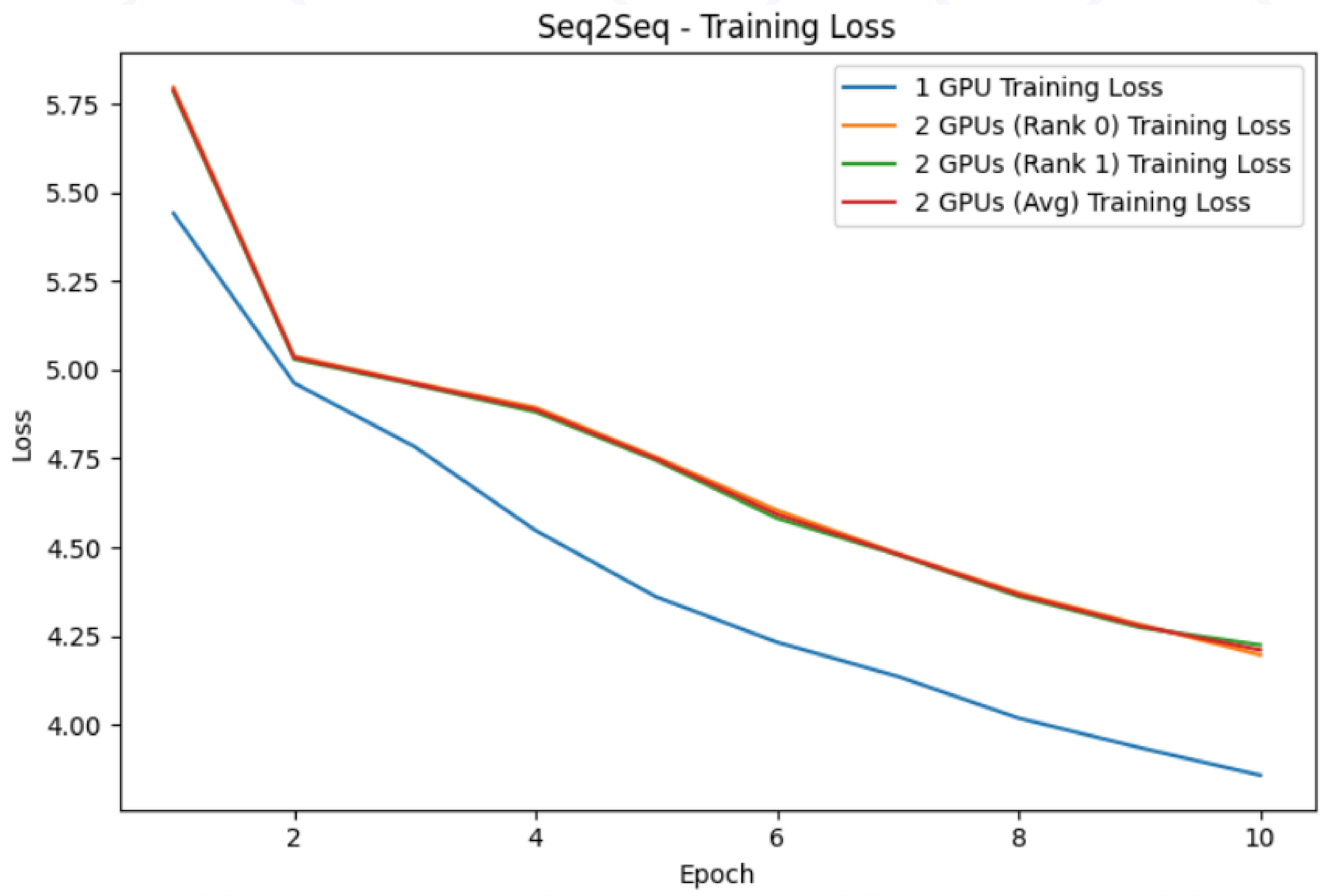


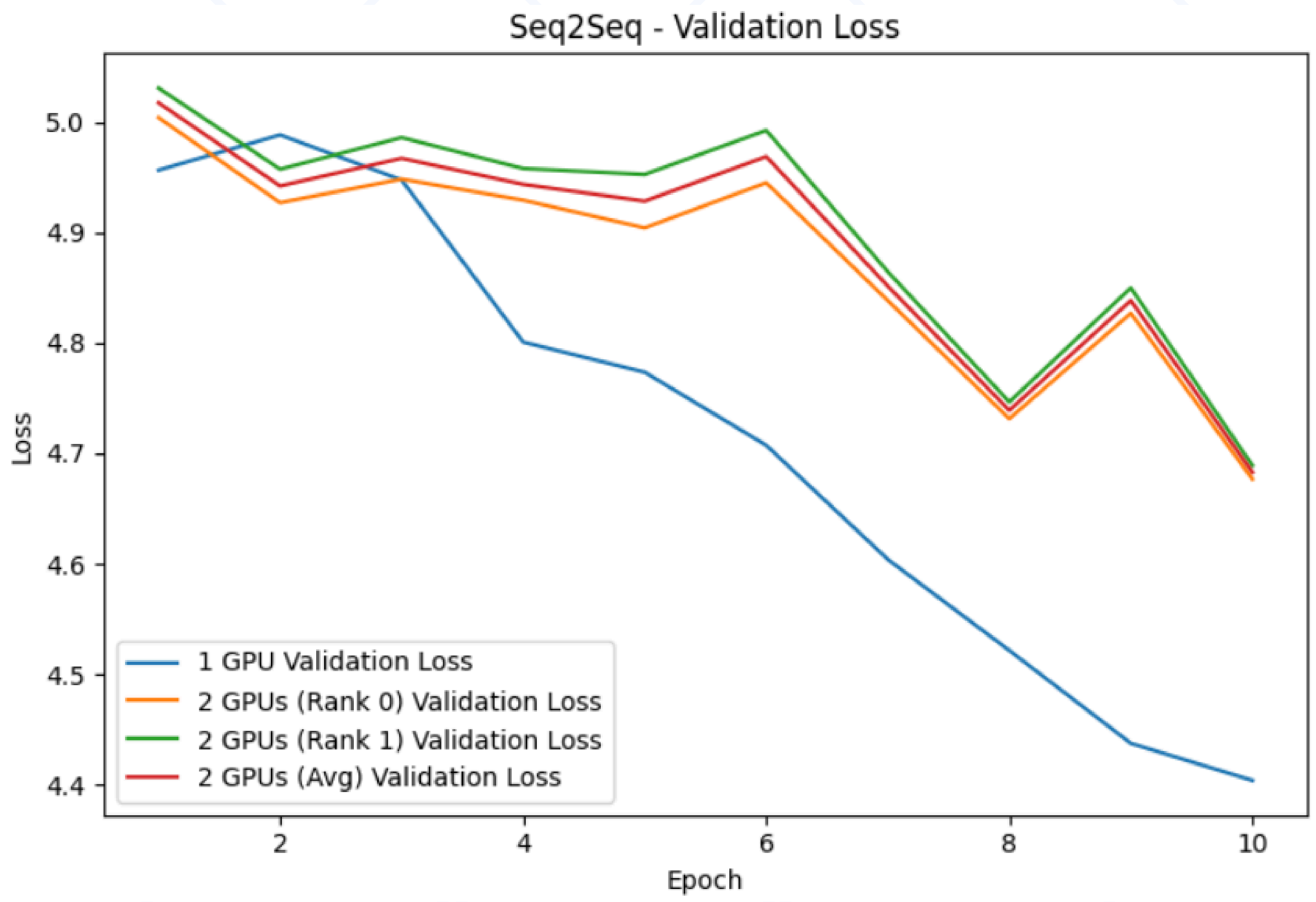


03

Seq 2 Seq

Training and Validation Loss

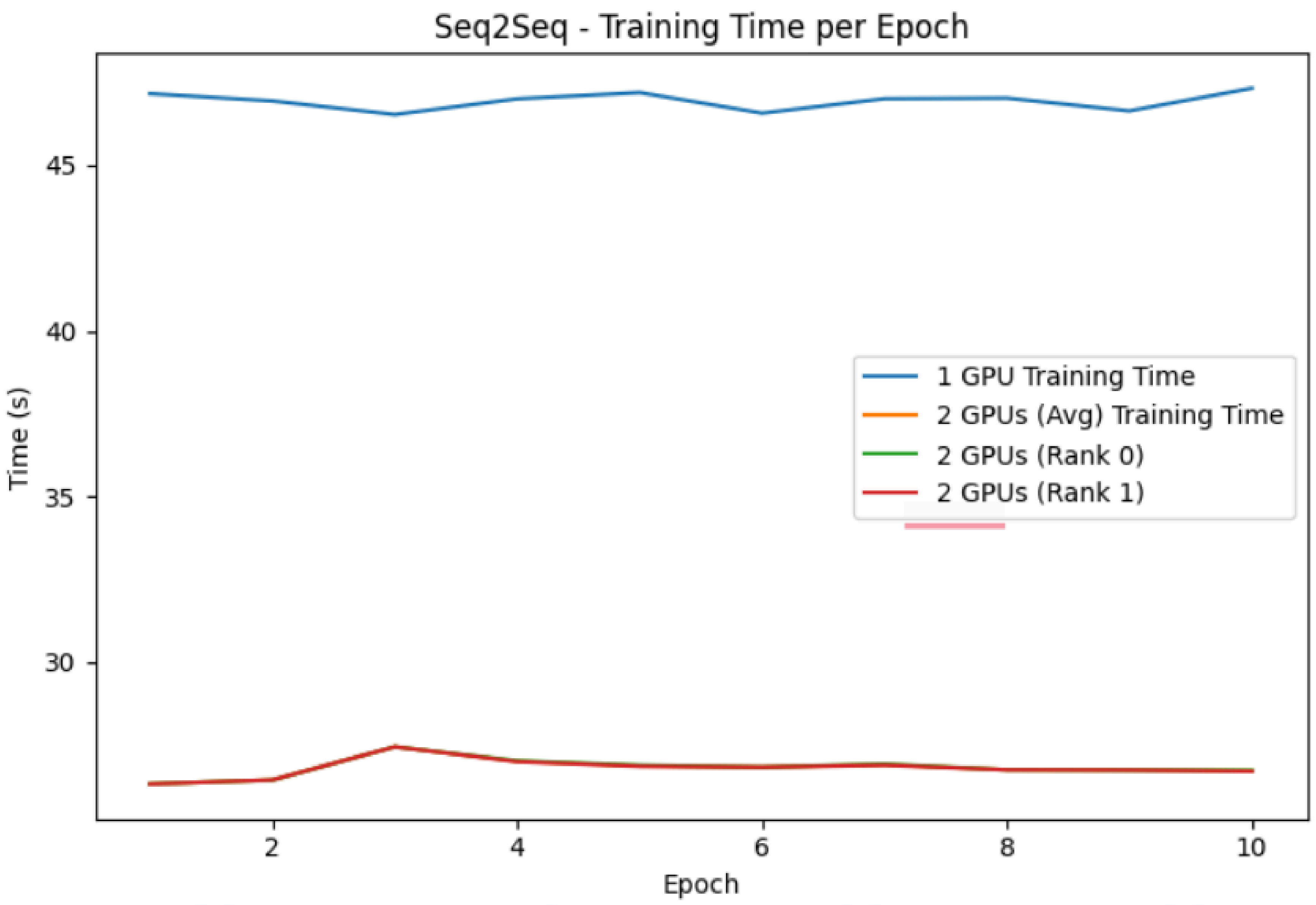


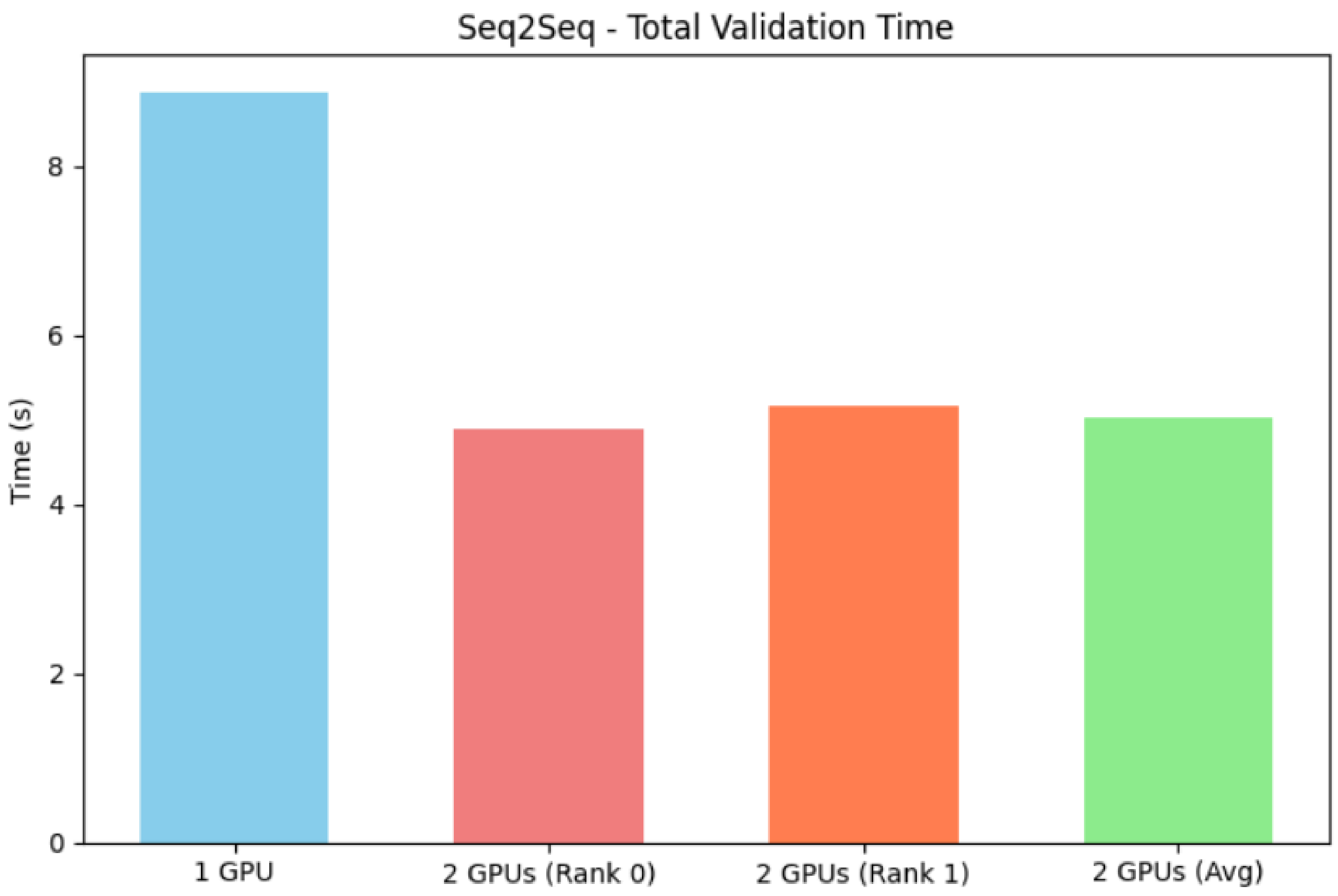


03

Seq 2 Seq

Training and Validation Time

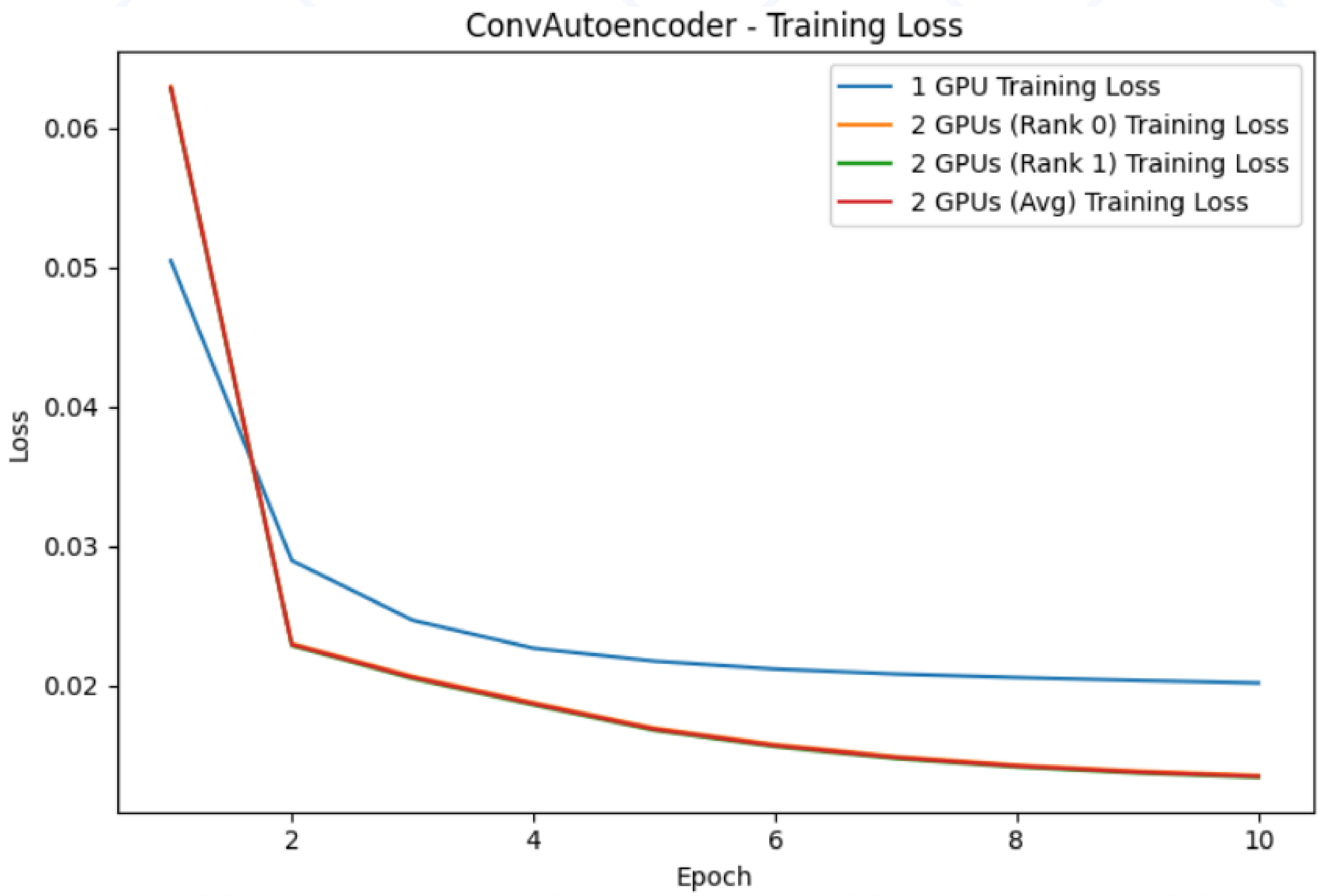


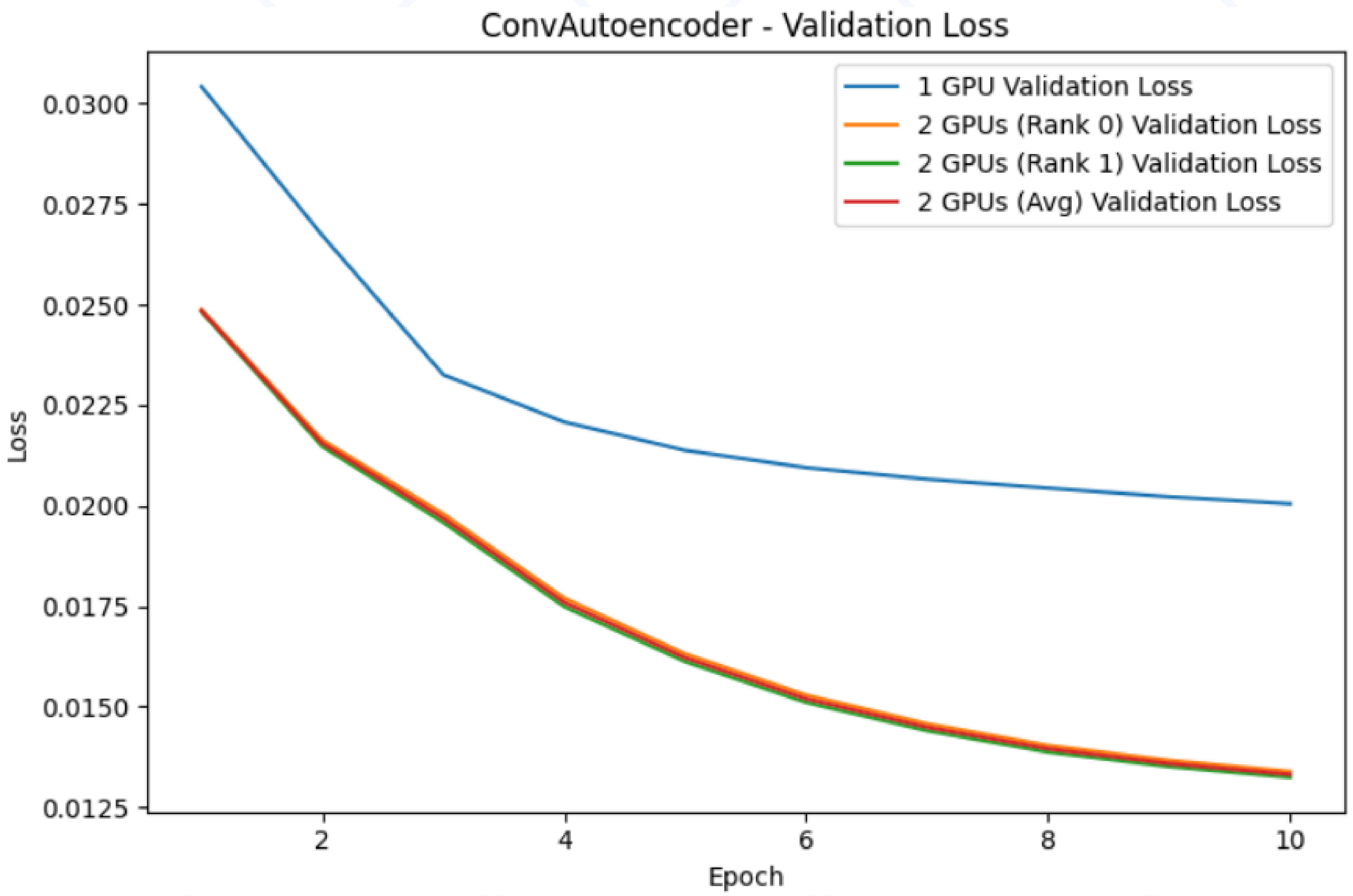


03

Convolutional Autoencoder

Training and Validation Loss

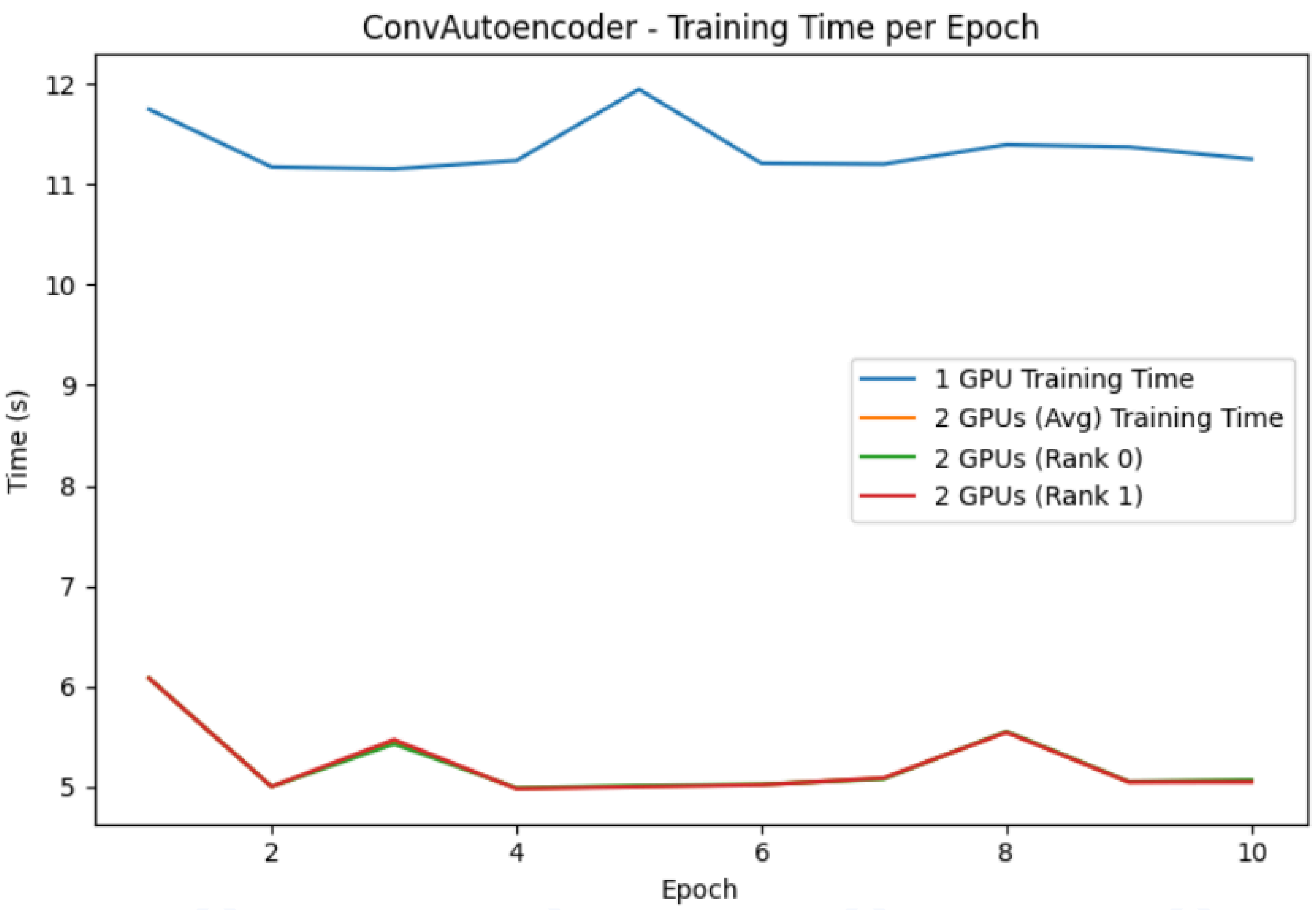


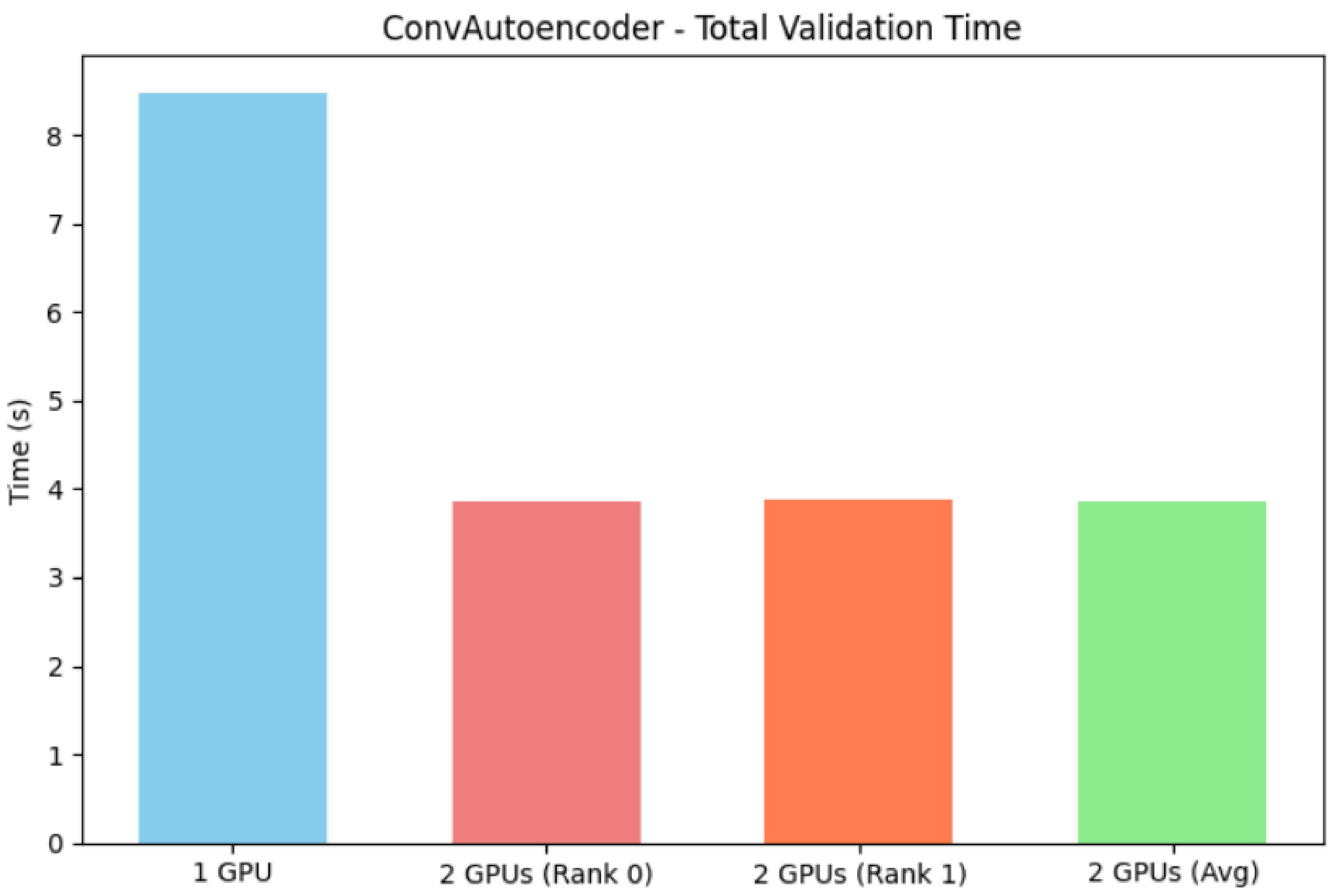


03

Convolutional Autoencoder

Training and Validation Time





04

Results Analysis

04

Results Analysis

Metric	Training and Validation Accuracy	Training and Validation Loss	Training and Validation Time
Distributed improves the metric	VGG11, MLP (Hybrid Parallelism)	VGG11, MLP (Hybrid Parallelism)	VGG11, NBoW, Seq2Seq, ConvAutoencoder, CNN, MLP (Data Parallelism)
Distributed has no change on the metric	NBoW, Seq2Seq, ConvAutoencoder	NBoW, Seq2Seq, ConvAutoencoder	-
Distributed deteriorates the metric	CNN	CNN	-

05

Conclusion

Thank You