Chasing Efficiency: What Happens When You Train $BERT_{BASE}$ with Parallel Architectures?

Daniel Gebre

Task Introduction: Fine-Tuning BERT Base Uncased

- **Choose of Model and Dataset**
  - **BERT** is a language model based on transformer architecture introduced by researchers at Google in 2018
  - **Optimized for Generatability:** Designed to process lower-case English text, is chosen for its ability to generalize well across different types of text.
  - **Squad v2 Dataset:** A widely respected benchmark dataset which provides a respected benchmark with realistic challenges, perfect for improving question-answering models
- **Objective of the Experiment**
  - The objective of this experiment was to compare the efficiency and effectiveness of different parallel training architectures in fine tuning BERT base uncased on Squad v2 Dataset using advanced GPU setups.
  - Computing resource: Single node machine with 4 NVIDIA A-100 GPUs, each 40 GB ram.
- **Conducted Experiments**
  - **Baseline:** Naïve Data Parallelism (Splits data across GPUs and Synchronizes gradients)
  - **Strategy -1 (Daniel):** Data - Pipeline Parallelism with *Deepspeed*
    - ***4 GPUs set up:*** Splits the BERT model into ***four*** sequential segments (stages)
    - ***2 GPUs set up:*** Splits the BERT model into ***two*** sequential segments (stages)

Understanding $BERT_{BASE}$ Architecture Components

1. Transformer Encoder

- **12 Layers**: Processes text bidirectionally to capture deep context. Each Encoder has:
  - ✓ **Multi-Head Self-Attention:** Focuses on relevant parts of the input for each word.
  - ✓ **Feed-Forward Networks:** Following the self-attention component, each layer has an FNN of size 3072

2. Input Embeddings

- **Token Embeddings:** Maps words to vectors.
- **Segment Embeddings:** Differentiates parts of the input.
- **Position Embeddings**: Marks word order in the sentence.

3. Model Capacity

- **110 Million Trainable Parameters:** Powers complex language understanding.

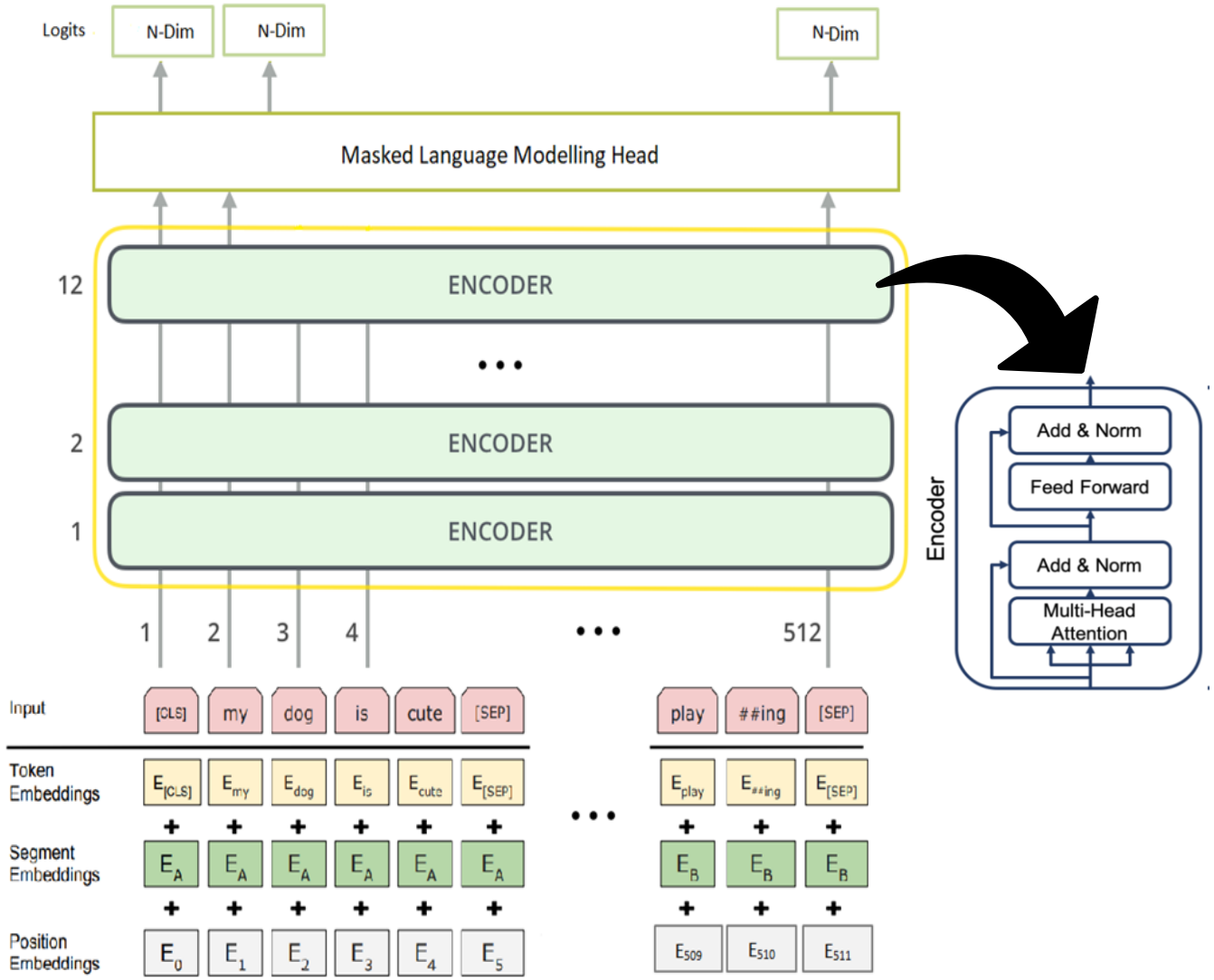


Fig 1. **$BERT_{BASE}$** - Model Architecture

Source: Fine-Tuning BERT

Baseline Training: Naïve Data Parallelism

Training Methodology

- **Library used: Pytorch**
- **Single Model Instance:** Runs one instance of BERT base uncased across all 4 GPUs.
- **Data Parallelism:** By wrapping the model in DDP, it automatically handles the distribution of data across the available GPUs (each GPU running a part of the batch).

```
self.train_sampler = DistributedSampler(self.train_dataset, num_replicas=world_size, rank=rank, shuffle=True)
self.train_loader = DataLoader(self.train_dataset, batch_size=16, sampler=self.train_sampler)

self.model = BertForQuestionAnswering.from_pretrained(self.model_name).to(self.device)
self.model = DDP(self.model, device_ids=[rank])
```

- **Gradient Synchronization**: During the backward pass, when loss.backward() is called, DDP ensures that gradients from the loss computation on each GPU are averaged across all GPUs

```
loss.backward()
optimizer.step()
```

- **Optimizer:** AdamW with specific hyperparameters

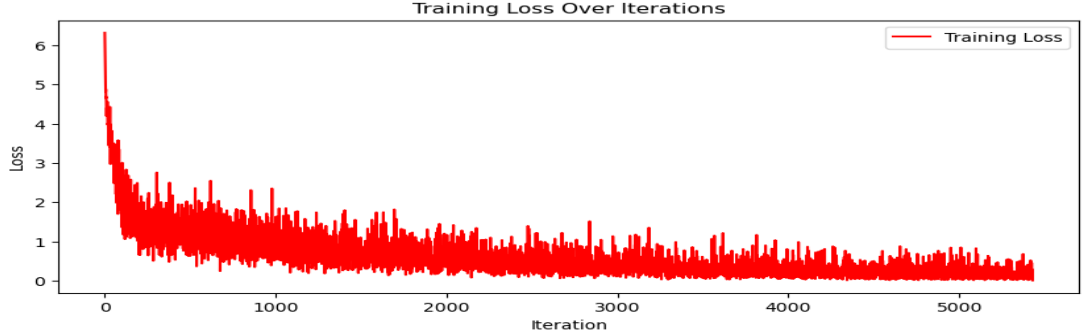
Baseline Parameters:

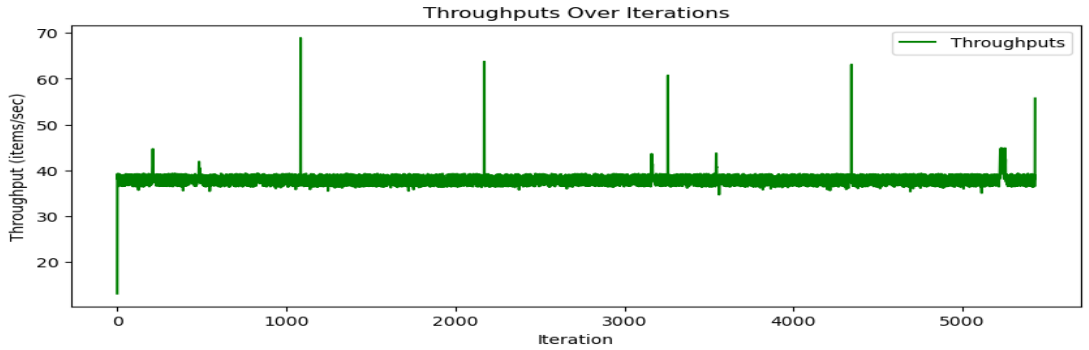
- **Batch size**: Fixed 16 examples
- **Micro batch:** 4 examples/GPU
- **Training Steps:** 5 Epochs – 5430 iterations

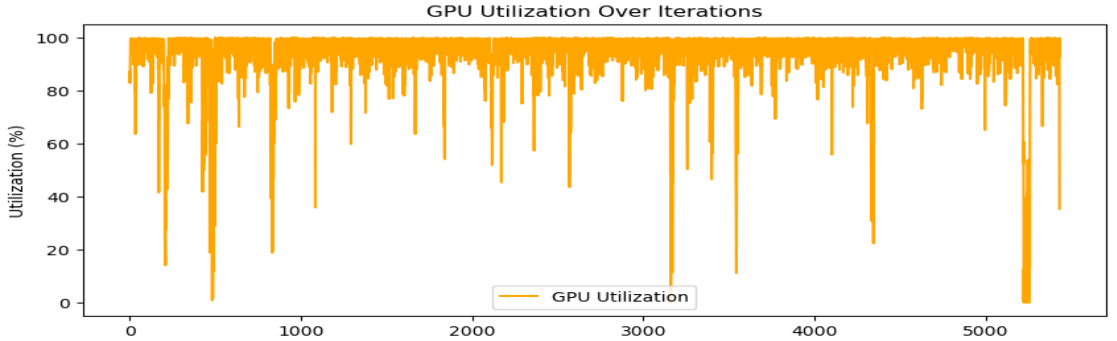
جامعة محمد بن زايد للذكاء الاصطناعي
MOHAMED BIN ZAYED UNIVERSITY OF ARTIFICIAL INTELLIGENCE

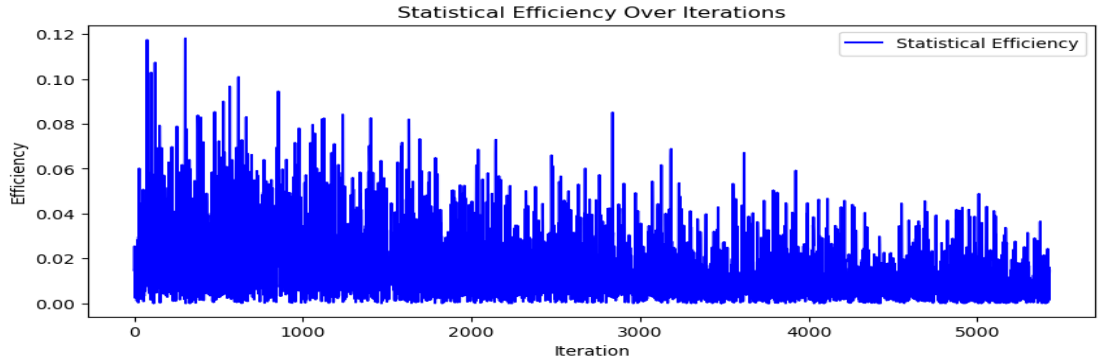
Baseline Performance

Runtime (sec or min)	**3688 sec 62 min**
Average Throughput (samples/sec)	**37.98**
Average Validation Loss	**0.6027**
Average GPU Utilization	**94.96 %**
Average Stats Efficiency	**0.01627**
Average Goodput	**0.6179**









جامعة محمد بن زايد للذكاء الاصطناعي
MOHAMED BIN ZAYED UNIVERSITY OF ARTIFICIAL INTELLIGENCE

Data-Pipeline Parallelism using DeepSpeed

General Training Methodology

- **Library used:** DeepSpeed and PyTorch
- **Model Instance**: BERT Base Uncased distributed across multiple GPUs.
- **Hybrid Parallelism**: Combining Data and Pipeline Parallelism for enhanced training throughput and efficiency.
- **Gradient Accumulation:** Set to 1, indicating immediate optimizer updates without accumulating gradients
- **Mixed Precision:** Enabled for faster computation and reduced memory usage.
- **Optimizer:** AdamW with specific hyperparameters aligned with DeepSpeed settings.
- **Batch size**: 16 examples
- **Micro batch:** set to "auto" for dynamic adjustment.
- **Epochs:** 5 Epochs

Experiment-1: 2 GPUs and Two Pipeline Stages

- **Pipeline Stages:** Two stages corresponding to the number of GPUs.
- **Training Steps:** 6735 iterations

Experiment-2: 4 GPUs and 4Pipeline Stages

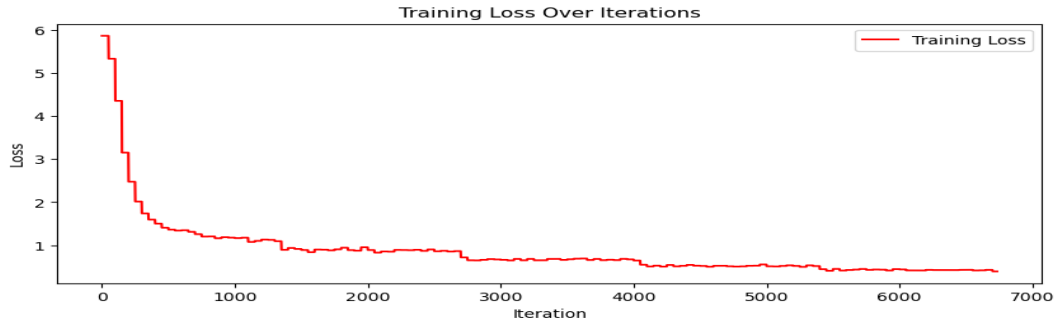
- **Pipeline Stages:** Four stages corresponding to the number of GPUs.
- **Training Steps:** 3340 iterations

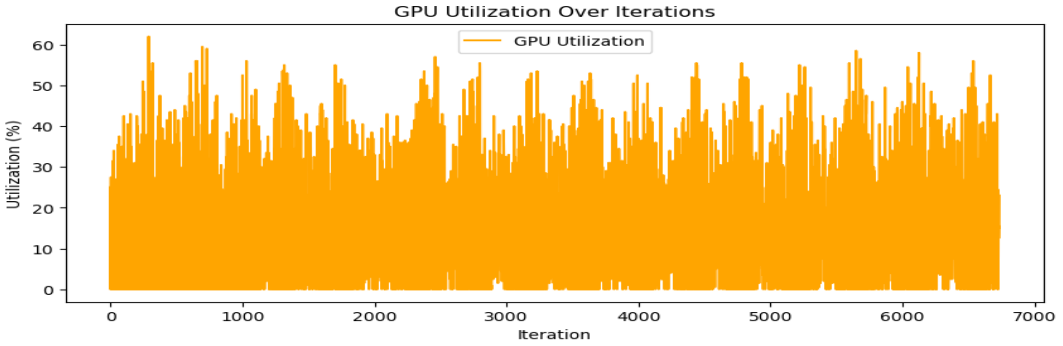
```
et main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
    model = BertForQuestionAnswering.from_pretrained('bert-base-uncased')
    model.to(device)  # Ensure model is on the correct device

    dataset = SquadDataset('./dataset/train-v2.0.json', tokenizer)

    training_args = TrainingArguments(
            output_dir='./results',                 # Directory for saving
            num_train_epochs=5,                     # Total number of trai
            per_device_train_batch_size=16,         # This will be dynamic
            gradient_accumulation_steps=1,          # Matches the DeepSpee
            learning_rate=3e-5,                     # Learning rate setting
            warmup_steps=500,                       # Warm-up steps used i
            weight_decay=0.01,                      # Weight decay to matc
            adam_epsilon=1e-6,                      # Epsilon for the Adam
            fp16=True,                              # Enable mixed precisi
            deepspeed='ds_config.json',             # Path to the DeepSpee
            logging_dir='./logs',                   # Directory for storin
            logging_steps=50,                       # Log every 50 steps.
            save_steps=1000,                        # Save the model every
            evaluation_strategy="steps",            # Evaluation strategy
            eval_steps=500                          # Evaluate the model e
        )
```

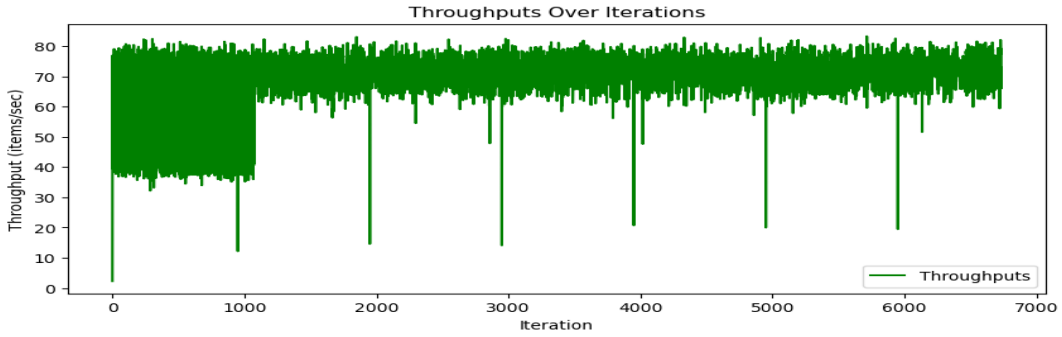
Pipeline Parallelism Performance

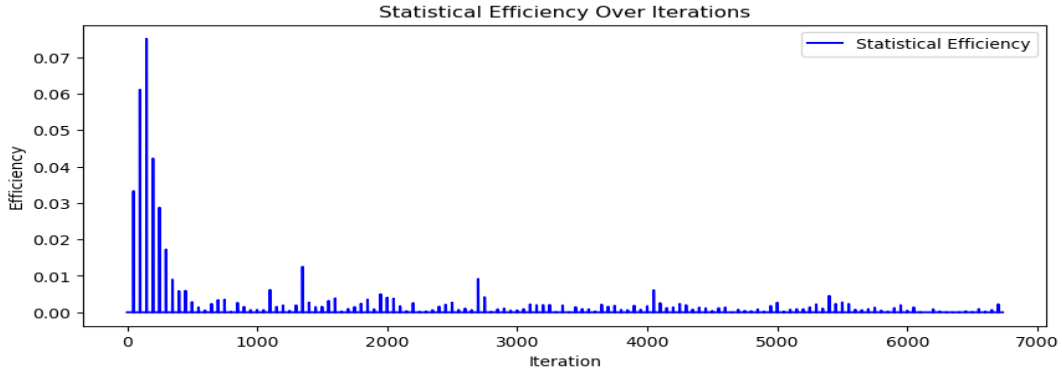
Experiment-1: 2 GPUs 2 Stages

Runtime (sec or min)	**3148 sec 52.46 min**	↓ 14.64 %
Average Throughput (samples/sec)	**70.40**	↑ 85.36%
Average Validation Loss	**0.8701**	↑ 44.37%
Average GPU Utilization	**14.45 %**	↓ 84.78 %
Average Stats Efficiency	**0.0000707**	↓ 99.57%
Average Goodput	**0.0049**	↓ 99.21%







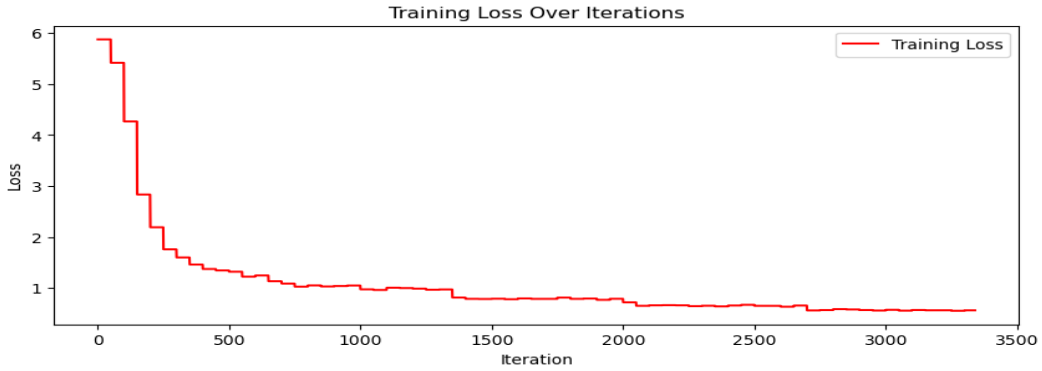


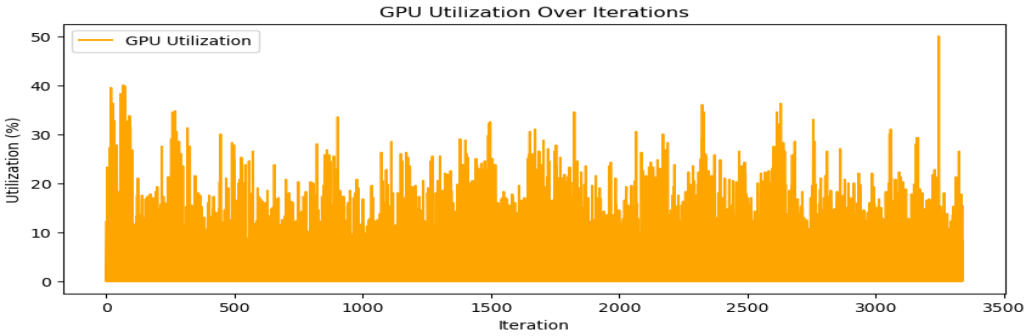
جامعة محمد بن زايد للذكاء الاصطناعي
MOHAMED BIN ZAYED UNIVERSITY OF ARTIFICIAL INTELLIGENCE

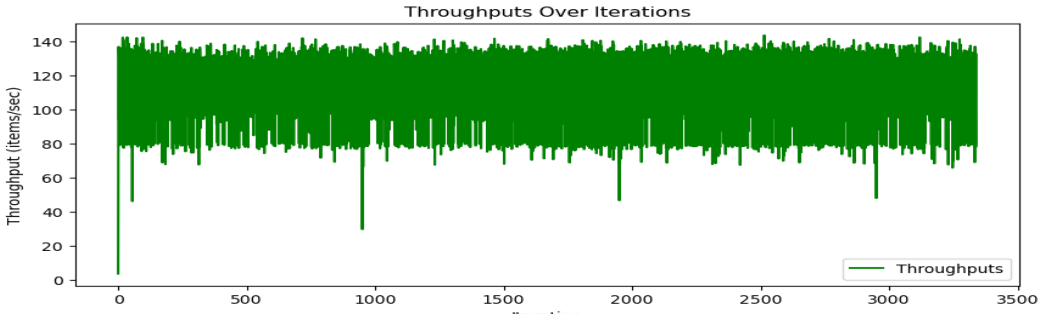
Pipeline Parallelism Performance

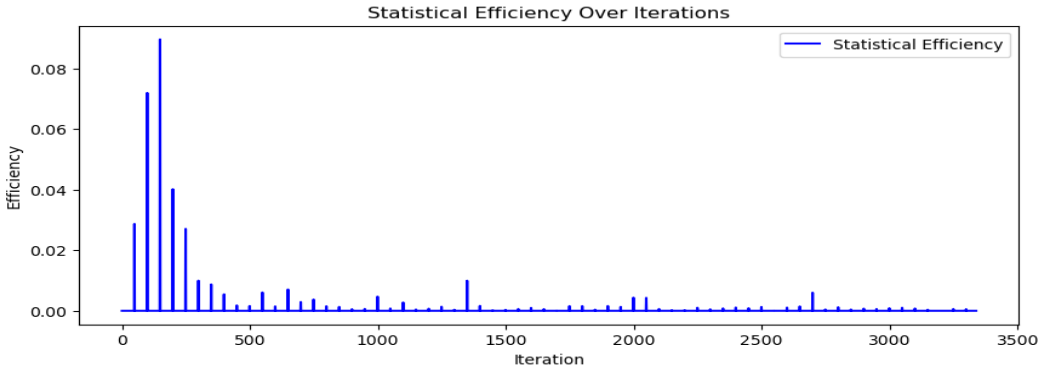
Experiment-2: 4 GPUs 4 Stages

Runtime (sec or min)	**1930.19 sec 32.17 min**	↓ 47.66 %
Average Throughput (samples/sec)	**116.27**	↑ 206.13 %
Average Validation Loss	**1.090**	↑ 80.85 %
Average GPU Utilization	**6.7454 %**	↓ 92.90 %
Average Stats Efficiency	**0.0001116**	↓ 99.31%
Average Goodput	**0.0129**	↓ 97.91%









Pipeline Parallelism Performance

Detailed Comparison against Baseline

Experiment	Runtime (sec)	Throughput (samples/sec)	Validation Loss	GPU Utilization (%)	Stats Efficiency	Goodput
Baseline	3688 sec	37.98	0.6027	94.96%	0.01627	0.6179
Pipeline 2 GPUs 2 Stages	3148 sec (↓14.64%)	70.40 (↑85.36%)	0.8701 (↑44.37%)	14.45% (↓84.78%)	0.0000707 (↓99.57%)	0.0049 (↓99.21%)
Pipeline 4 GPUs 4 Stages	1930.19 sec (↓47.66%)	116.27 (↑206.13%)	1.090 (↑80.85%)	6.7454% (↓92.90%)	0.0001116 (↓99.31%)	0.0129 (↓97.91%)

Experiment findings and discussion

- **Runtime Efficiency:** Reduced runtime in the "Pipeline 4 GPUs 4 Stages" indicates a more efficient processing, suitable for time-sensitive tasks.
- **Throughput Maximization:** "Pipeline 4 GPUs 4 Stages" achieved the highest throughput, beneficial for handling large datasets swiftly.
- **Validation Loss Concern:** Despite efficiency gains, an increase in validation loss suggests potential overfitting or inadequate training in more complex models.
- **Statistical Efficiency & Goodput:** The significant decrease in stats efficiency and goodput in both pipeline experiments raises concerns about the effectiveness of resource utilization and data processing quality

Questions??