Programming Assignment report

Domain : Distributed training/ Parallel optimization

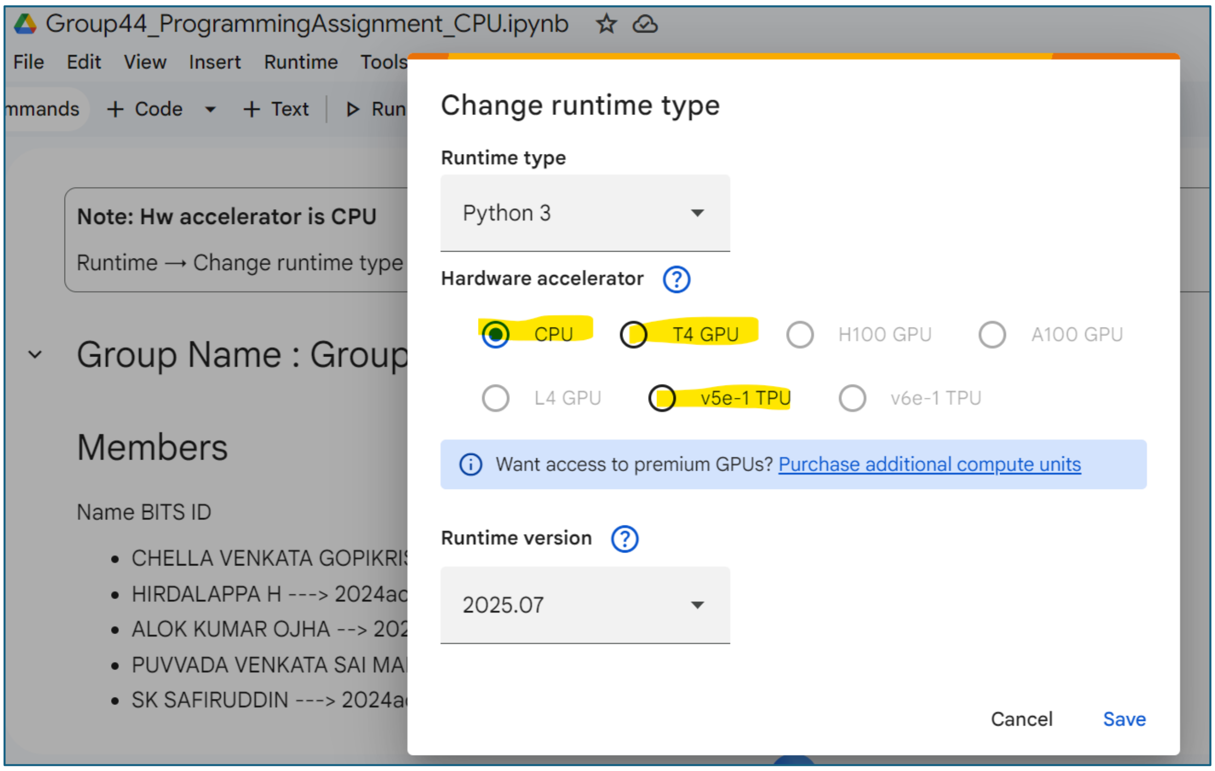
Research Paper: TDML (Trustworthy Distributed Machine Learning) Framework

Group Name : Group 44	
Name	**BITS ID**
HIRDALAPPA H	2024ac05306
CHELLA VENKATA GOPIKRISHNA	2024ac05978
ALOK KUMAR OJHA	2024ad05055
PUVVADA VENKATA SAI MANOJ CHANDRA	2024ac05227
SK SAFIRUDDIN	2024ac05781

Contents

1 HW Accelerator

- Runtime → Change runtime type
- Under Hardware accelerator, select: GPU or TPU
- Save and reconnect



Group44_ProgrammingAssignment_CPU.ipynb

Note: Hw accelerator is CPU

Runtime → Change runtime type

Group Name : Group

Members

Name BITS ID

- CHELLA VENKATA GOPIKRI
- HIRDALAPPA H ---> 2024ac
- ALOK KUMAR OJHA --> 202
- PUVVADA VENKATA SAI MA
- SK SAFIRUDDIN ---> 2024a

Change runtime type

Runtime type: Python 3

Hardware accelerator: [✓ CPU] T4 GPU, H100 GPU, A100 GPU, L4 GPU, v5e-1 TPU, v6e-1 TPU

Want access to premium GPUs? Purchase additional compute units

Runtime version: 2025.07

Figure 1: Google Colab hardware (HW) accelerators

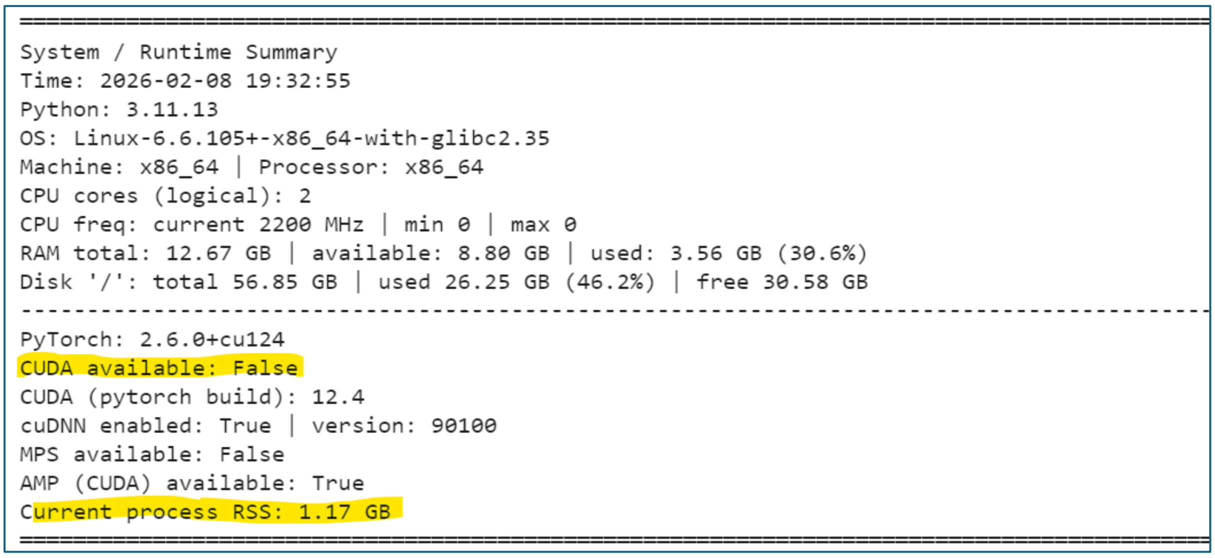
2 Impact of HW accelerator on the system

A **hardware accelerator** (such as a GPU, TPU, FPGA, ASIC, or DSP) is a specialized processing unit designed to execute specific classes of computations significantly faster and more energy-efficiently than a general-purpose CPU. By offloading compute-intensive tasks from the CPU, hardware accelerators fundamentally alter system behavior, performance characteristics, and overall system design.

The integration of hardware accelerators impacts a system across multiple dimensions, including **computational performance, architectural design, power consumption, software stack complexity, cost efficiency, and scalability**. General-purpose CPUs, massively parallel GPUs, and machine-learning-optimized TPUs each introduce distinct system-level trade-offs due to their differing design philosophies and execution models.

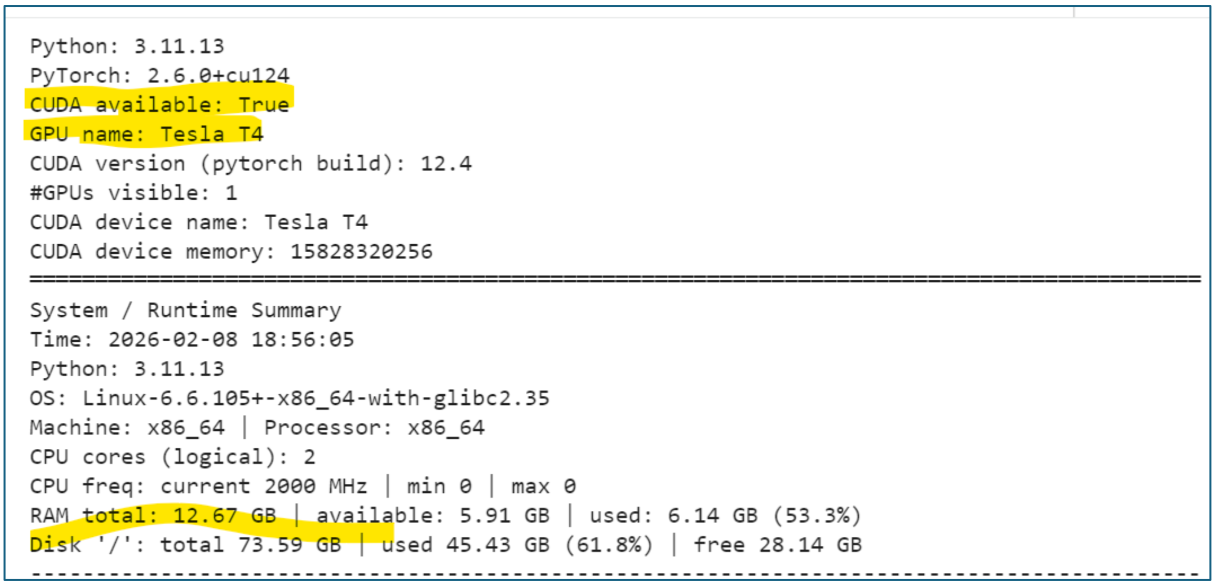
To evaluate these impacts, we conducted experiments by executing the same workload on **CPU, GPU, and TPU platforms**. During these experiments, we observed and analyzed key system performance metrics, including **workload selection, batch size configuration, data loader behavior, and the relationship between training epochs and execution time**. These metrics provide insight into how different accelerators influence throughput, latency, and overall training efficiency.

The results and observations from these experiments are discussed in detail in the following sections, with a focus on comparative system performance across CPU-, GPU-, and TPU-based execution environments.



```
==========================================================================================
System / Runtime Summary
Time: 2026-02-08 19:32:55
Python: 3.11.13
OS: Linux-6.6.105+-x86_64-with-glibc2.35
Machine: x86_64 | Processor: x86_64
CPU cores (logical): 2
CPU freq: current 2200 MHz | min 0 | max 0
RAM total: 12.67 GB | available: 8.80 GB | used: 3.56 GB (30.6%)
Disk '/': total 56.85 GB | used 26.25 GB (46.2%) | free 30.58 GB
------------------------------------------------------------------------------------------
PyTorch: 2.6.0+cu124
CUDA available: False
CUDA (pytorch build): 12.4
cuDNN enabled: True | version: 90100
MPS available: False
AMP (CUDA) available: True
Current process RSS: 1.17 GB
==========================================================================================
```

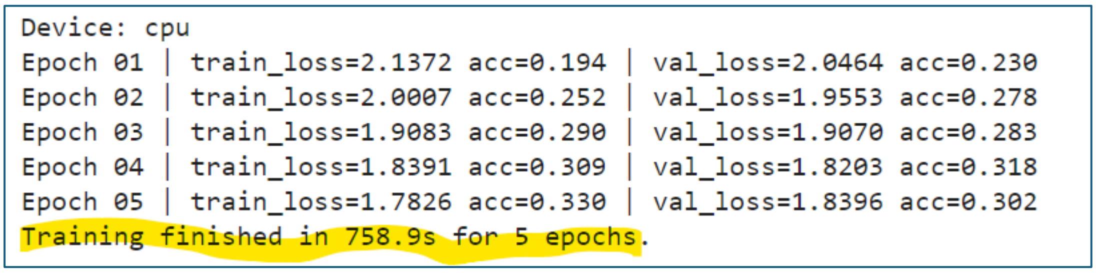
Figure 2: CPU accelerator (CUDA → False)



```
Python: 3.11.13
PyTorch: 2.6.0+cu124
CUDA available: True
GPU name: Tesla T4
CUDA version (pytorch build): 12.4
#GPUs visible: 1
CUDA device name: Tesla T4
CUDA device memory: 15828320256
==========================================================================================
System / Runtime Summary
Time: 2026-02-08 18:56:05
Python: 3.11.13
OS: Linux-6.6.105+-x86_64-with-glibc2.35
Machine: x86_64 | Processor: x86_64
CPU cores (logical): 2
CPU freq: current 2000 MHz | min 0 | max 0
RAM total: 12.67 GB | available: 5.91 GB | used: 6.14 GB (53.3%)
Disk '/': total 73.59 GB | used 45.43 GB (61.8%) | free 28.14 GB
------------------------------------------------------------------------------------------
```

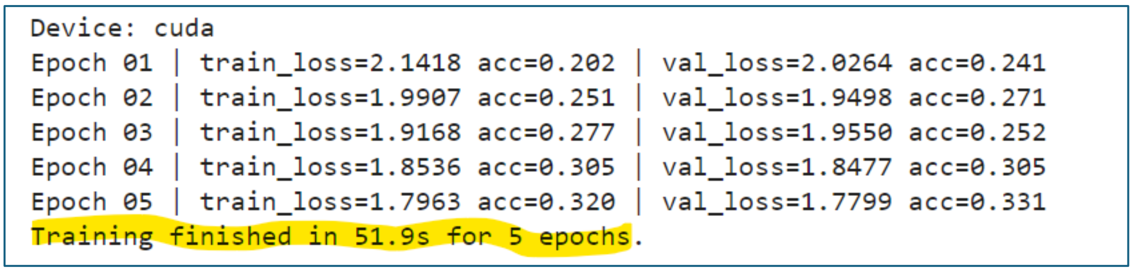
Figure 3: GPU accelerator (CUDA → True)

3 Impact of training time



```
Device: cpu
Epoch 01 | train_loss=2.1372 acc=0.194 | val_loss=2.0464 acc=0.230
Epoch 02 | train_loss=2.0007 acc=0.252 | val_loss=1.9553 acc=0.278
Epoch 03 | train_loss=1.9083 acc=0.290 | val_loss=1.9070 acc=0.283
Epoch 04 | train_loss=1.8391 acc=0.309 | val_loss=1.8203 acc=0.318
Epoch 05 | train_loss=1.7826 acc=0.330 | val_loss=1.8396 acc=0.302
Training finished in 758.9s for 5 epochs.
```

Figure 4: CPU training timing



```
Device: cuda
Epoch 01 | train_loss=2.1418 acc=0.202 | val_loss=2.0264 acc=0.241
Epoch 02 | train_loss=1.9907 acc=0.251 | val_loss=1.9498 acc=0.271
Epoch 03 | train_loss=1.9168 acc=0.277 | val_loss=1.9550 acc=0.252
Epoch 04 | train_loss=1.8536 acc=0.305 | val_loss=1.8477 acc=0.305
Epoch 05 | train_loss=1.7963 acc=0.320 | val_loss=1.7799 acc=0.331
Training finished in 51.9s for 5 epochs.
```

Figure 5: GPU Training timing

The GPU (CUDA) significantly outperforms the CPU, completing 5 epochs in **51.9 s** compared to **758.9 s** on the CPU, indicating a ~14× speed-up.
This demonstrates that GPUs greatly reduce training time while maintaining similar convergence behavior, making them more suitable for compute-intensive workloads.

4 Impact of Selecting num_workers and batch_size on System Performance

- CPU → smaller batch size, moderate workers
- GPU → larger batch size, higher workers (balanced with CPU/I/O)
- TPU → very large batch size, minimal manual worker tuning

Thus, accelerator choice directly determines how batch_size and num_workers should be configured for optimal system performance.

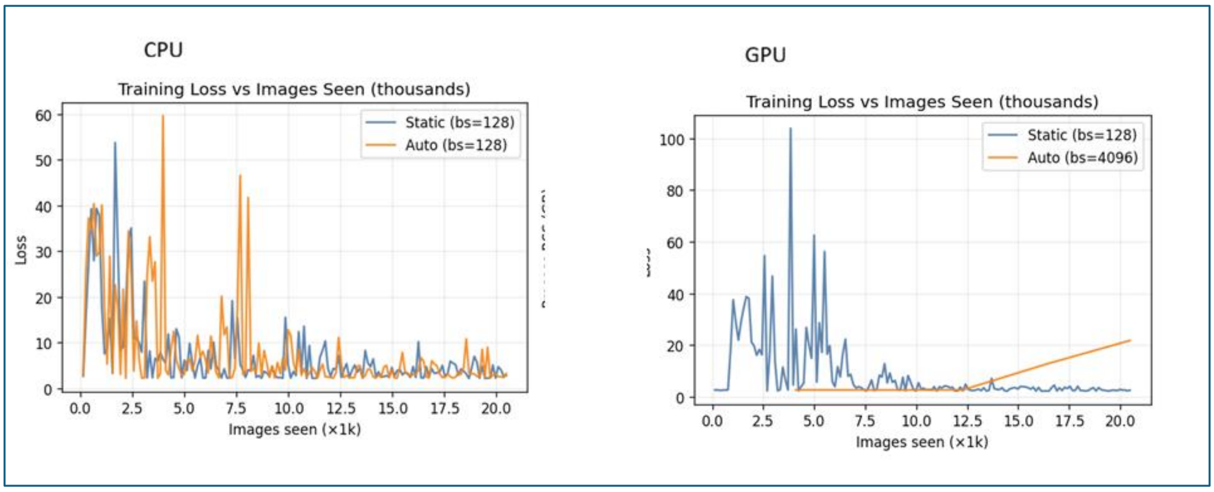


Figure 6: Training loss vs Image size

On the **CPU**, static and auto batch sizes (**both bs=128**) show similar convergence behavior, indicating limited benefit from dynamic batch sizing due to constrained parallelism.
On the **GPU**, auto batch sizing selects a much larger batch (**bs=4096**), significantly improving hardware utilization and resulting in smoother, more stable loss reduction as more images are processed.
This shows that **auto/large batch sizes are far more effective on GPUs**, while CPUs gain minimal benefit due to limited parallel execution capability.

5 Impact on Throughput and Peak Memory

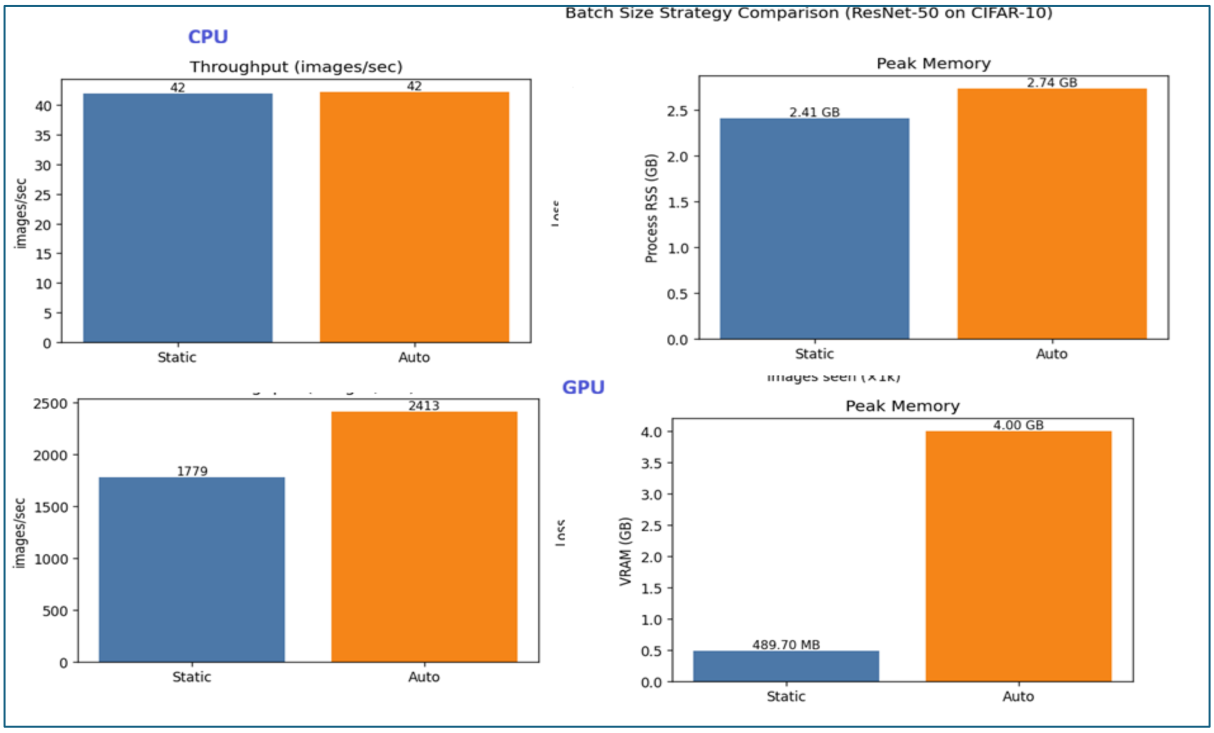


Figure 7: Batch size selection & Peak memory

Inference from the Batch Size Strategy Comparison (CPU vs GPU):
On the **CPU**, auto batch sizing provides **no throughput gain** over static batching (~42 images/sec) while slightly increasing memory usage, indicating limited parallelism benefits.
On the **GPU**, auto batch sizing significantly improves throughput (**~1779 → 2413 images/sec**) by better utilizing GPU parallelism, but at the cost of much higher VRAM usage (**~0.5 GB → 4 GB**).

- This shows that **auto/large batch sizes are highly effective on GPUs**, whereas CPUs see minimal performance improvement but increased memory overhead.

6 Performance: Time, Throughput, Speedup, Efficiency

CPU selection:

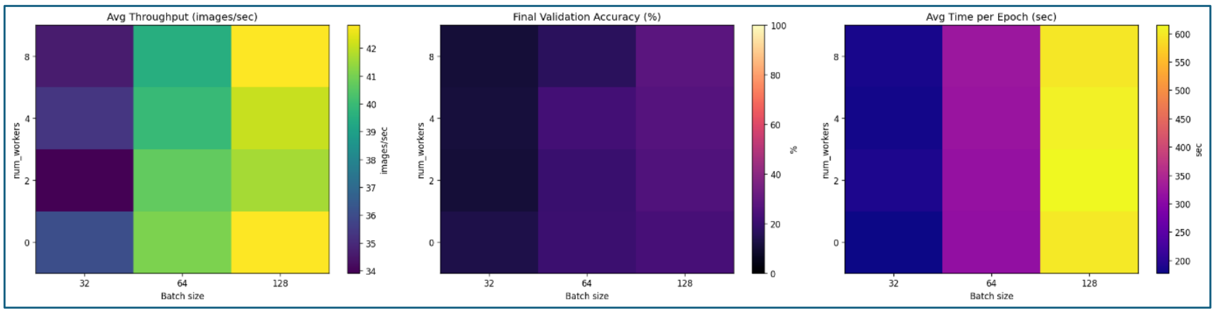


Figure 8: Throughput vs Epochs (CPU)

GPU selection:

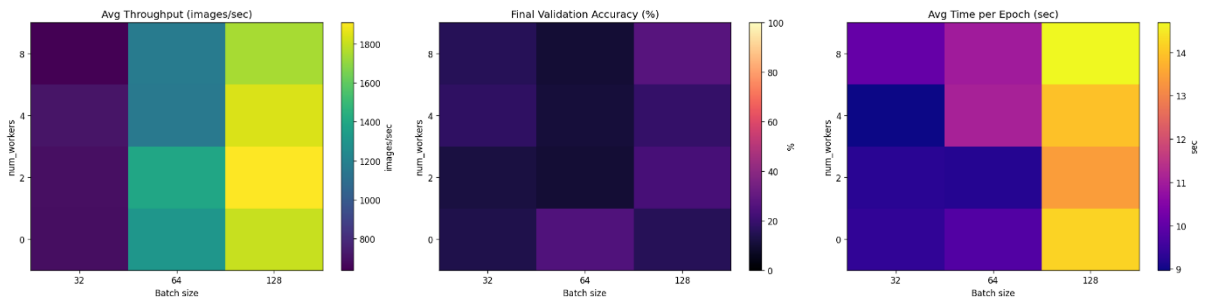


Figure 9:Throughput vs Epochs (GPU)

On the **CPU**, increasing batch size slightly improves throughput but significantly increases time per epoch, while accuracy remains largely unaffected; higher num_workers offers only marginal gains due to limited parallelism.
On the **GPU**, larger batch sizes dramatically improve throughput and reduce epoch time, showing effective parallel utilization, with accuracy remaining stable across configurations.
Overall, GPUs benefit strongly from larger batch sizes and moderate workers, whereas CPUs show diminishing returns from aggressive tuning of both parameters.

7 Single vs Pipeline vs Federated learning(FL)

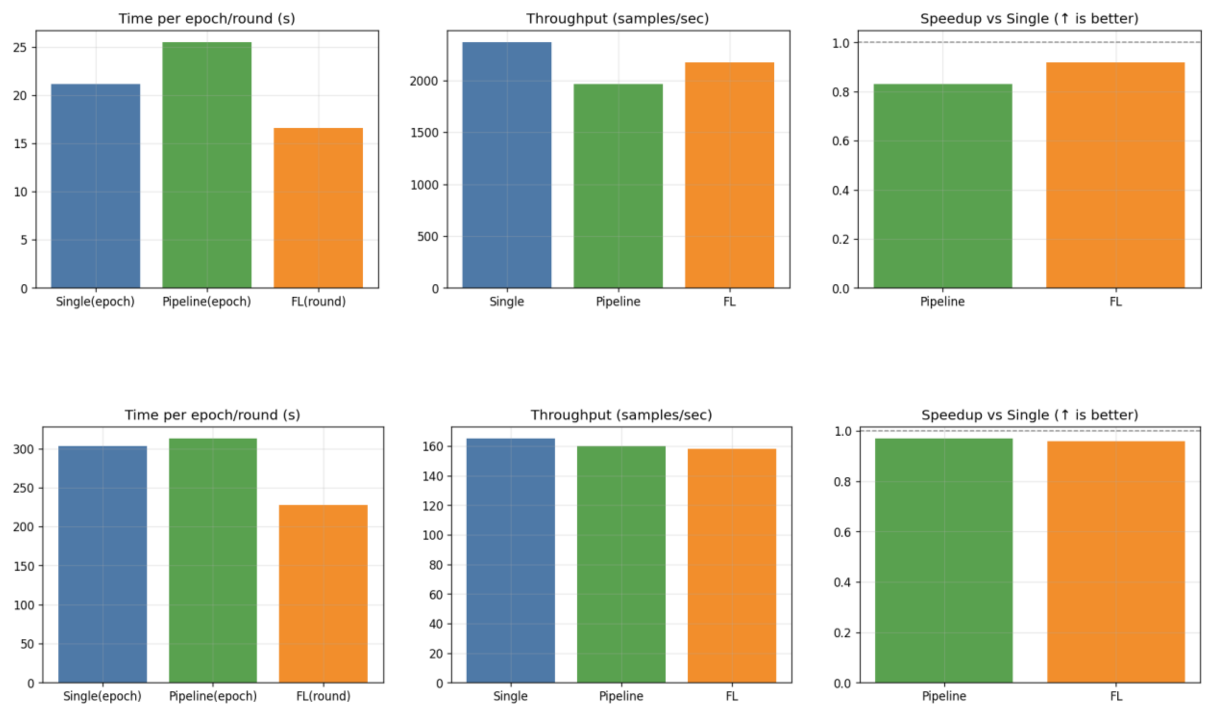


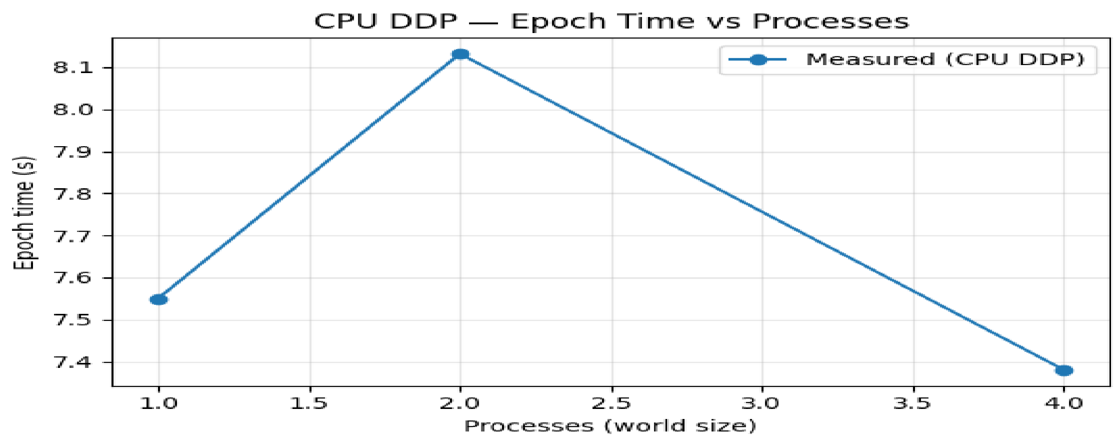
Figure 10: Single vs Pipeline vs FL

On the **GPU**, Federated Learning (FL) achieves the **lowest time per round** and higher throughput than pipeline execution, indicating better overlap and utilization, though neither exceeds single-node speed due to coordination overhead.
On the **CPU**, both pipeline and FL show **limited or no speedup** over single execution, with reduced throughput caused by communication and scheduling overheads.
Overall, **GPU platforms benefit more from FL and pipelining**, while CPUs suffer from overheads that offset parallelism gains.

- **On the GPU**, single training achieves the highest throughput, while pipelining suffers from pipeline bubbles (~20%) that reduce efficiency despite slightly higher accuracy.
- **Federated Learning (FL)** shows lower throughput than single training and fails to converge (accuracy ≈0.10), indicating aggregation or configuration issues.
- **On the CPU,** both pipelining and FL provide little to no speedup over single execution, as coordination and scheduling overhead dominate.
- Overall, **advanced parallel strategies benefit GPUs more than CPUs**, but their effectiveness depends heavily on proper configuration and workload balance.

8 CPU DDP — Epoch Time vs Processes

CPU selected:



GPU selected:

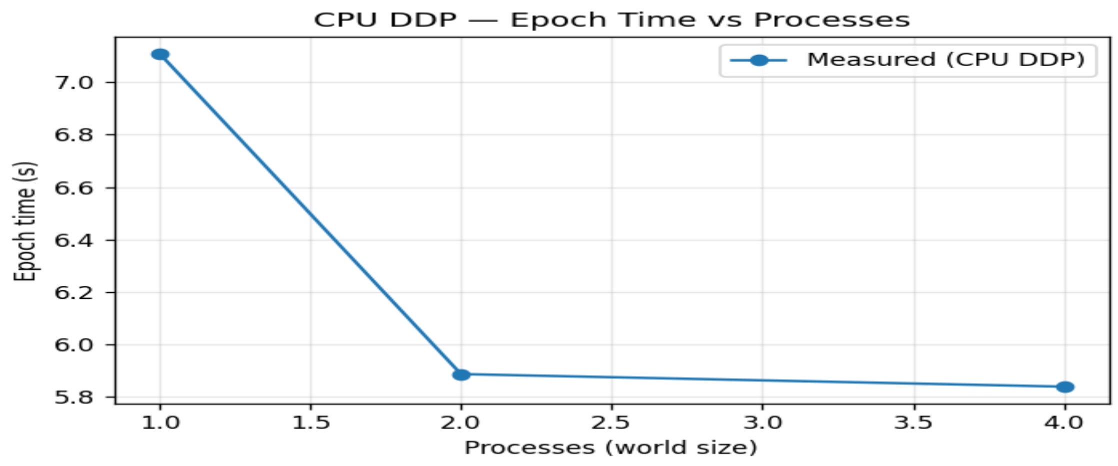


Figure 11: Epoch vs number of processors

Increasing the number of processes from 1 to 2 reduces epoch time, showing some benefit from data parallelism on the CPU.
However, scaling beyond 2 processes provides **diminishing or inconsistent gains**, with communication and synchronization overhead offsetting parallelism.
The non-monotonic behavior indicates that **CPU DDP is limited by inter-process overhead and memory bandwidth**, making it less scalable than GPU-based DDP.

9 Conclusion:

Hardware accelerator choice has a significant impact on system performance and scalability. GPUs deliver the best performance gains with appropriate batch sizes and parallel strategies, while CPUs show limited scalability due to communication and resource contention. Advanced techniques like pipelining and FL are effective only when properly tuned and are more beneficial on GPU-based systems.