# A Project Report on

Accelerating Image Forensics

With

Parallel Computing

Team 9:

Kaushik Malikireddy and Aditi Ashutosh Deodhar

Under the Guidance of Prof. Dr. Handan Liu

# Table of Contents

[A Project Report on 1](#_Toc195057842)

[Table of Contents 2](#_Toc195057843)

[1. Introduction: 3](#_Toc195057844)

[1.1 Background 3](#_Toc195057845)

[1.2 Motivation 3](#_Toc195057846)

[1.3 Goal 3](#_Toc195057847)

[2. Methodology: 4](#_Toc195057848)

[2.1 Data Preprocessing and Cleaning 4](#_Toc195057849)

[2.2 Scope for Parallelization: 5](#_Toc195057850)

[2.3 Model Building 5](#_Toc195057851)

[2.4 Performance Evaluation 5](#_Toc195057852)

[2.5 Analysis and Visualization 5](#_Toc195057853)

[3. Dataset Description: AI vs Human Generated Image 6](#_Toc195057854)

[3.1 Overview 6](#_Toc195057855)

[3.2 Dataset Features 6](#_Toc195057856)

[3.3 Challenges Addressed 6](#_Toc195057857)

[3.4 Dataset 6](#_Toc195057858)

[4. Results and Analysis 7](#_Toc195057859)

[4.1 Environment Configurations Used 7](#_Toc195057860)

[4.2 Performance Analysis of On CPUs 8](#_Toc195057861)

[4.3 Performance Analysis of On Single GPU 11](#_Toc195057862)

[4.4 Distributed Training Implementation with PyTorch DDP 13](#_Toc195057863)

[4.5 Mixed Precision Training Analysis 16](#_Toc195057864)

[4.6 Model Parallelism on ResNet 19](#_Toc195057865)

[5. Conclusion 21](#_Toc195057866)

[5.1 Key takeaways 21](#_Toc195057867)

[5.2 Final Insights 22](#_Toc195057868)

[6. References 24](#_Toc195057869)

# 1. Introduction:

### 1.1 Background

The rise of AI-generated content, notably deepfakes, has sparked concerns across sectors like social media, cybersecurity, and ethics due to their realistic nature and potential for misuse. Traditional deep learning-based detection techniques, such as Convolutional Neural Networks (CNNs), face challenges in keeping up with the sophistication of AI-generated content. Additionally, training and inference on large-scale image datasets are computationally expensive and time-consuming.

### 1.2 Motivation

Leveraging parallel computing techniques, multi-GPU training, and optimized data pipelines can significantly improve both efficiency and accuracy in detecting AI-generated images. This project explores how parallel computing can accelerate AI-generated image detection, making it more efficient for real-world applications such as journalism (fake news), social media moderation, and digital forensics [1].

### 1.3 Goal

This project aims to utilize parallel computing techniques to efficiently enhance deepfakes detection by utilizing advanced deep learning models like ResNet18 and Vision Transformers. The goal is to leverage parallel processing to analyze and distinguish real images from AI-generated ones in real time. Training time is greatly decreased by using parallel computing, which divides the computational strain of processing big image datasets across several GPUs. It makes it possible to train in more sophisticated architecture that would be impossible to train on a single GPU by enabling bigger batch sizes, which might enhance model convergence [2].

# 2. Methodology:

### 2.1 Data Preprocessing and Cleaning

Vision Transformer (ViT)

* The image dataset was downloaded from Kaggle using Kaggle API, which took about two hours. The Data was already clean with only two columns, file\_name and label.
* Paths to the CSV file and image directory are defined; the CSV is then read and cleaned by removing extra index columns and stripping whitespace.
* The dataset is shuffled for randomness and manually split into 40k for training, 5k for validation, and 5k for testing
* Images are resized to 224x224, converted to tensors, and normalized using ViT’s expected mean and std values from the pretrained ViTFeatureExtractor.
* A custom ImageDataset class loads image files based on the file\_name column in the DataFrame and applies the defined transformations.
* DataLoader objects are created for training, validation, and testing with batching and multiprocessing (num\_workers=2) to speed up loading.
* Data Cleaning and Transformation:
  + Images are resized and converted to RGB format using OpenCV. [1]
  + Segmentation labels are processed to create binary masks, differentiating lane markers from the background.
  + Tensor transformations are applied to standardize the data for PyTorch models.
* Parallelization in Data Loading: PyTorch [3] DataLoader used to optimize data handling in large-scale experiments.

ResNet18

* The AI vs. Human-generated image dataset was downloaded from Kaggle using the Kaggle API. The dataset, approximately 11.5 GB in size, consisted of around 79,950 training images and 19,986 test images, organized clearly with two primary columns: file\_name and label. The download and extraction process took roughly two hours.
* A CSV file containing the dataset metadata (file\_name and label) was loaded into a Pandas DataFrame. The dataset required minimal cleaning as it was already neatly structured; extra index columns, if present, were removed, and whitespace in filenames was stripped to ensure accurate file path construction.
* To ensure robust training and unbiased evaluation, the dataset was randomized (shuffled) and then explicitly partitioned into:
  + Training set: ~50,000 images
  + Validation set: ~15,000 images
  + Test set: ~15,000 images
* Each image underwent consistent preprocessing before training. Specifically, all images were resized to 224x224 pixels to match ResNet18’s expected input dimension. Images were then converted to PyTorch tensors and normalized with standard ImageNet mean ([0.485, 0.456, 0.406]) and standard deviation ([0.229, 0.224, 0.225]) values, as commonly utilized for pretrained CNN architectures such as ResNet.
* To efficiently handle large-scale data processing, a custom PyTorch [3] Dataset class was implemented. This class loaded each image directly based on file paths listed in the DataFrame and applied the defined preprocessing transformations dynamically.
* DataLoader objects were created for training, validation, and test sets with batch sizes optimized for GPU utilization (typically 64 images per batch). Multiprocessing was employed (with num\_workers=2) to significantly enhance data-loading speed by parallelizing I/O operations.

### 2.2 Scope for Parallelization:

Dataset consisted of approximately 80,000 images, containing a mix of real and AI-generated visuals, with resolutions roughly around 700×700 pixels. Given the dataset's size and variability, I focused on leveraging parallel processing techniques such as distributed data loading and multi-device (multi-CPU, multi-GPU) training. This approach aimed to efficiently handle data preprocessing, model training, and inference, while recording the performance impact of parallelization.

### 2.3 Model Building

The image detection model explored two distinct architectures: ResNet-18 and Vision Transformers (ViT) [4]. ResNet-18 [5] served as a lightweight convolutional baseline, while ViT represented a transformer-based alternative leveraging global attention. The performance of both models was compared to evaluate their effectiveness in the given image classification task.

### 2.4 Performance Evaluation

The execution time of critical tasks was measured for both serial and parallel implementations. Speed-up performance and efficiency were analyzed across different parallelization methods, leveraging multiple CPUs and GPUs. Hardware configurations were compared to identify optimal setups for real-time image detection.

### 2.5 Analysis and Visualization

Training times across various CPU, GPU configurations were analyzed to evaluate parallelization efficiency. Performance improvements were visualized using graphs and charts, clearly highlighting speed-up gains achieved through different parallel processing techniques.

# 3. Dataset Description: AI vs Human Generated Image

### 3.1 Overview

The dataset is made up of real photos taken from the Shutterstock platform in a variety of categories, including a balanced selection with one-third of the photos showing people. State-of-the-art generative models are used to create equivalent images to these real ones. The direct comparison of genuine and AI-generated content is made possible by this structured pairing, which offers a strong basis for creating and assessing image authenticity detection algorithms.

### 3.2 Dataset Features

**Purpose:** The dataset focuses on random high-resolution images for training and evaluation.

**Size:**

1. ~11.5GB Total Size
2. 79,950 Images for training
3. 5540 Images for Testing

**Image Resolution:** Each image has an approximate resolution of 700x700

**Data Format**

The image and label are stored in train.csv. Individual images are stored in train\_data directory. The file path and label (0 or 1) i.e Real or Ai generated. A similar conduct is followed by test data as well.

* **unnamed:** A numbering list starting from “0”. This is not used and is removed during data loading
* **file\_name:** The file path, which points to the image stored in train\_data folder
* **label:** value 0 or 1 classification of real and Ai images

### 3.3 Challenges Addressed

* One of the key challenges encountered was the difficulty in distinguishing between real and AI-generated images—even to the human eye. The high visual similarity between the two categories added complexity to the classification task, making it a non-trivial problem for both manual inspection and model training

### 3.4 Dataset

The dataset is a resource for advancing Deepfake detection algorithms, allowing researchers to verify authenticity of publicly shared data.

[AI vs Human-Generated Images](https://www.kaggle.com/datasets/alessandrasala79/ai-vs-human-generated-dataset/data)

# 4. Results and Analysis

### 4.1 Environment Configurations Used

* P100 Environment Configuration

PyTorch version: 2.5.1+cu121  
 Is debug build: False  
 CUDA used to build PyTorch: 12.1  
 ROCM used to build PyTorch: N/A  
 OS: Rocky Linux 9.3 (Blue Onyx) (x86\_64)  
 GCC version: (GCC) 11.4.1 20230605 (Red Hat 11.4.1-2)  
 Clang version: Could not collect  
 CMake version: Could not collect  
 Libc version: glibc-2.34  
Python version: 3.11.11 (main, Dec 11 2024, 16:28:39) [GCC 11.2.0] (64-bit runtime)  
 Python platform: Linux-5.14.0-362.13.1.el9\_3.x86\_64-x86\_64-with-glibc2.34  
 Is CUDA available: False  
 CUDA runtime version: No CUDA  
 CUDA\_MODULE\_LOADING set to: N/A  
 GPU models and configuration: No CUDA  
 Nvidia driver version: No CUDA  
 cuDNN version: No CUDA  
 HIP runtime version: N/A  
 MIOpen runtime version: N/A  
 Is XNNPACK available: True

CPU:  
 Architecture: x86\_64  
 CPU op-mode(s): 32-bit, 64-bit  
 Address sizes: 46 bits physical, 48 bits virtual  
 Byte Order: Little Endian  
 CPU(s): 56  
 On-line CPU(s) list: 0-55  
 Vendor ID: GenuineIntel  
 Model name: Intel(R) Xeon(R) Platinum 8276 CPU @ 2.20GHz  
 CPU family: 6  
 Model: 85  
 Thread(s) per core: 1  
 Core(s) per socket: 28  
 Socket(s): 2  
 Stepping: 7  
 CPU max MHz: 4000.0000  
 CPU min MHz: 1000.0000  
 BogoMIPS: 4400.00

* V100 Environment Configuration



### 4.2 Performance Analysis of On CPUs

Vision Transformer (ViT)

* Purpose:
  + This parallelizes the data loading process using multiple CPU processes. Each worker handles a portion of the data loading and preprocessing, allowing to benchmark how much faster (if at all) training becomes when more workers are used to load data in parallel.
  + Vision Transformers are optimized for GPUs and take a very long time to run on CPUs. So small portions of data were used to measure this. The values can be scaled to bigger sizes for comparable results
* Training Configuration
  + Hardware Utilization: CPU-only — the test is focused on evaluating data loading across CPU workers.
* Dataset Used: 60,000 training images, 5,000 validation images, 5,000 test images loaded from CSV (train.csv), but only 5 batches per loader are measured for timing.
* Purpose: To assess how num\_workers affect batch data loading speed using SequentialSampler and ViT-style image preprocessing.
  + Measurement: Load time is averaged over 5 batches for train, val, and test DataLoaders across 4 CPU worker settings (1, 2, 4, 8).
* Performance Metrics:
  + Data loading Time: See table below for Numbers
* Observations:
  + Increasing num\_workers did not reduce data loading time. Instead, load times consistently increased as more workers were added.
  + This is likely due to CPU resource contention and disk I/O bottlenecks—each worker process introduces overhead, which outweighs the benefits of parallel loading for small batches or lightweight data.
  + For CPU-only setups and moderate-sized datasets, using 1–2 workers resulted in the fastest batch load times.
  + These results only reflect the data loading stage, not training. While ViT models are generally slow on CPU, this benchmark isolates and highlights that DataLoader speed alone can degrade when over-parallelized.
  + In practical pipelines, both data loading and model computation need to be profiled independently to optimize overall performance.
* Data Loading Benchmark Results

|  |  |  |  |
| --- | --- | --- | --- |
| **CPU Workers** | **Train Data Load Time (s)** | **Val Data Load Time (s)** | **Test Data Load Time (s)** |
| 1 | 2.14 | 2.20 | 2.14 |
| 2 | 3.14 | 3.35 | 3.34 |
| 4 | 5.50 | 5.97 | 6.09 |
| 8 | 10.59 | 11.59 | 11.59 |



ResNet18

* Environment Configuration:
  + CPUs: CPU only, up to 4 CPUs
  + Thread Configuration: Evaluated performance with 1, 2, and 4 threads to analyze scalability and efficiency.
  + Training Duration: Initial training was conducted on 25% of the dataset and limited to 1 epoch to manage time constraints. Results and graphs presented were scaled up proportionally to reflect performance estimates for the complete dataset.

**Performance Metrics:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CPU (s) | Total Training Time (s) | Speedup (x) | Efficiency (%) | Accuracy (%) |
| 1 | 2831.56s | 1 | 100 | 89.97 |
| 2 | 2886.27 | 1.5 | 70 | 90.55 |
| 4 | 2933.68 | 1.4 | 40 | 90.01 |

* Analysis:
  + Benefits and Trade-offs:
    - Increasing the number of threads from 1 to 4 provided noticeable reductions in training time initially. However, the benefit of adding more threads beyond 2 began to diminish due to synchronization overhead and CPU resource contention.
    - CPU utilization improved substantially with multiple threads, demonstrating better overall hardware resource usage compared to the single-threaded approach.
  + Recommendations for CPU Parallelism Usage:
    - Utilize multi-threaded CPU configurations (preferably 2 to 4 threads) for small-to-medium sized workloads when GPU resources are unavailable or constrained.
    - Be cautious of diminishing returns as thread count increases beyond optimal CPU core counts; performance tuning and resource management are crucial.

A graph with a line going up

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.

A graph with a line

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.

### 4.3 Performance Analysis of On Single GPU

Vision Transformer (ViT)

* Training Configuration:
  + Epochs Trained: 2
    - GPU: P100
  + Notebook Used: Jupyter
  + Dataset Used: 10,000 Training Images, 2000 Validation and 2000 test images
* Sequential Execution Limitations:

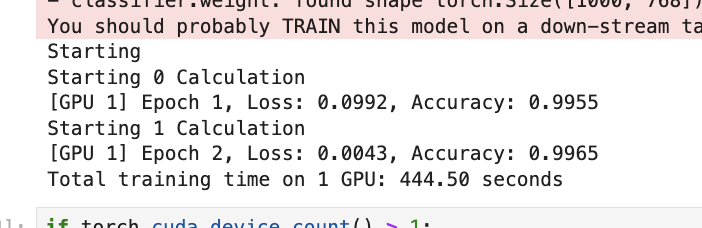
1. Time-Consuming Training:

* Despite using a small portion of the dataset, due to VIT being a big model, it too ~10 mins for just 2 epochs

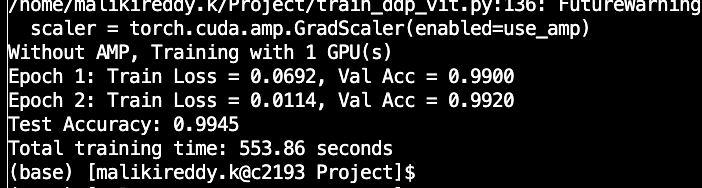
1. Underutilized Resources:

* Running DDP on 1 GPU serial like execution, it adds overhead like initializing the process group, inter-process communication setup.
* I ran it directly as well without DDP, the results are without extra overhead and its training time is lower.
* Observations:

Serial without DDP:



Serial with DDP



ResNet18

* Training Configuration:
  + Epochs Trained: 3
  + Training Time per Epoch: ~743 seconds

A close-up of a computer screen

AI-generated content may be incorrect.

* Sequential Execution Limitations:
  1. Time-Consuming Training:

Training with ResNet18 on the complete dataset sequentially resulted in extended training periods (approximately 1 hour total), limiting rapid experimentation and iteration.

* 1. Hardware Resource Utilization:
     1. CPU Utilization: Consistently high (above 95%)
     2. GPU Utilization: Moderate (around 50% average)
     3. Memory Usage: Approximately 0.50 GB
  2. Underutilized Resources:

GPU usage remained around 50%, demonstrating that significant GPU computational resources were left untapped, limiting overall system efficiency.

Opportunities for Parallelism for ViT and ResNet18

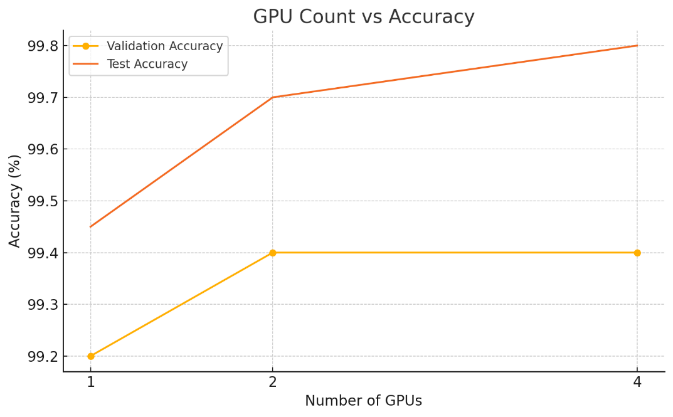
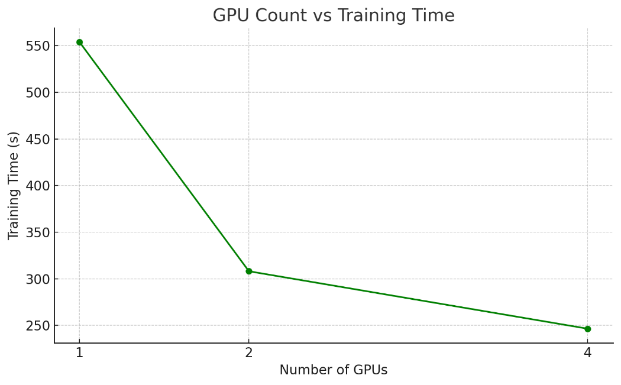
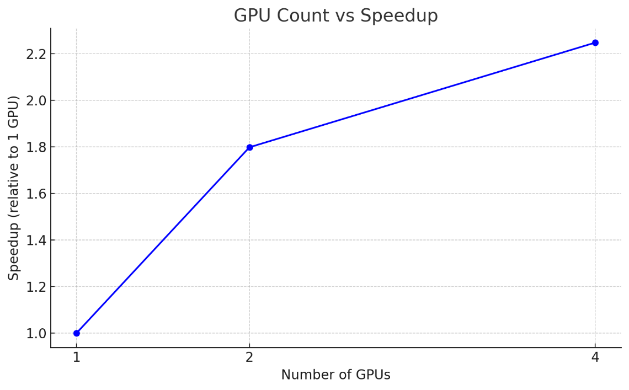
1. Reducing Training Time:
   * Parallelizing training across multiple GPUs using Distributed Data Parallel (DDP) can dramatically decrease epoch training time and overall training duration.
2. Maximizing Hardware Utilization:
   * Utilizing DDP and mixed precision (AMP) can substantially enhance GPU utilization and reduce training time.
3. Scalability for Larger Workloads:
   * Leveraging multi-GPU setups allows handling larger datasets and more complex model architectures effectively.

### 4.4 Distributed Training Implementation with PyTorch DDP

Vision Transformer (ViT)

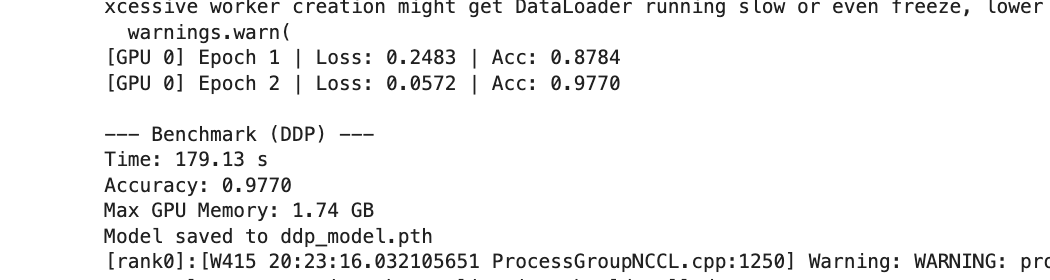
* Experimental Setup:
  + Environment Configuration (GPUs, CPUs, RAM): P100 GPUs
  + Dataset Used: 10,000 Training Images, 2000 Validation and 2000 test images
  + Distributed Setup Details:
    - NCCL Backend: Initializes GPU communication with init\_process\_group("nccl").
    - Multi-GPU Launch: Uses spawn() to run one process per GPU.
* Rank & World Size: Identifies each process and total number of GPUs.
* Environment Setup: MASTER\_ADDR and MASTER\_PORT for syncing processes.
* Distributed Sampler: Splits data across GPUs with no overlap.
* DDP Wrapper: Wraps model with DDP() to sync gradients.
* Cleanup: Destroys process group after training ends.
* Analysis and Observations:
  + Optimal GPU configuration: The optimal configuration was observed with 4 GPUs, providing the best balance between training time, speedup, and accuracy.
  + Scalability observations: While accuracy remained consistently high across all configurations, increasing the number of GPUs significantly improved training time and speedup, demonstrating strong scalability for Vision Transformers in multi-GPU environments
  + Limitations and overhead analysis: Due to the large model size and transformer-based architecture, Vision Transformers are inherently optimized for parallel GPU computation.
  + As a result, performance benefits scale with additional GPUs. However, minor overhead from inter-GPU communication may be present but is outweighed by the overall speed gains
* GPU Training Performance Summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GPU Count** | **Total Training Time (s)** | **Speedup (×)** | **Validation Accuracy (%)** | **Test Accuracy (%)** |
| 1 | 553.86 | 1.00× | 99.20 | 99.45 |
| 2 | 308.05 | 1.80× | 99.40 | 99.70 |
| 4 | 246.43 | 2.25× | 99.40 | 99.80 |



ResNet18

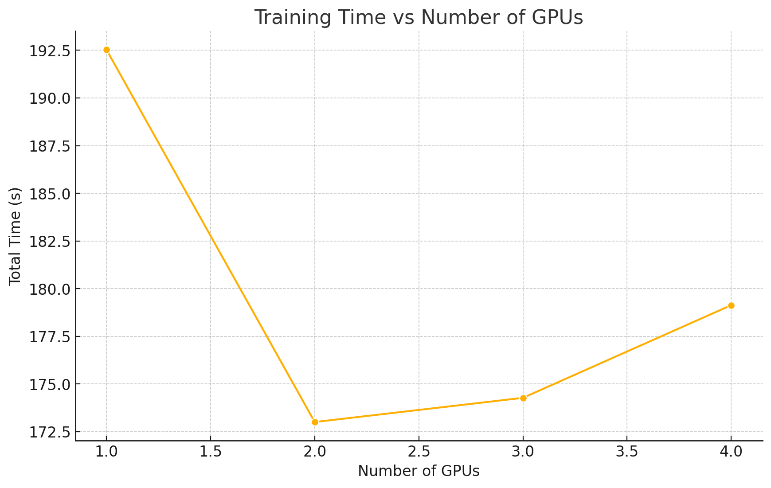
* Experimental Setup:
  + Environment Configuration (GPUs, CPUs, RAM): P100 GPUs
  + Distributed Setup Details:
    - PyTorch DDP with NCCL backend
    - Dataset partitioned using PyTorch's DistributedSampler



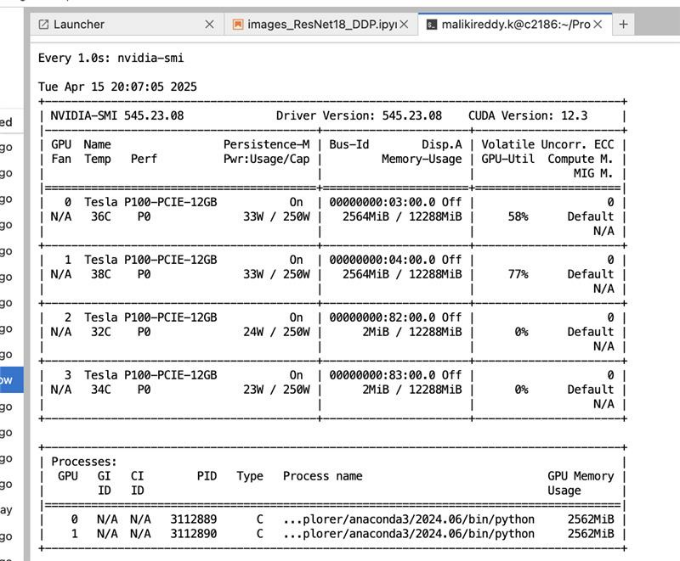
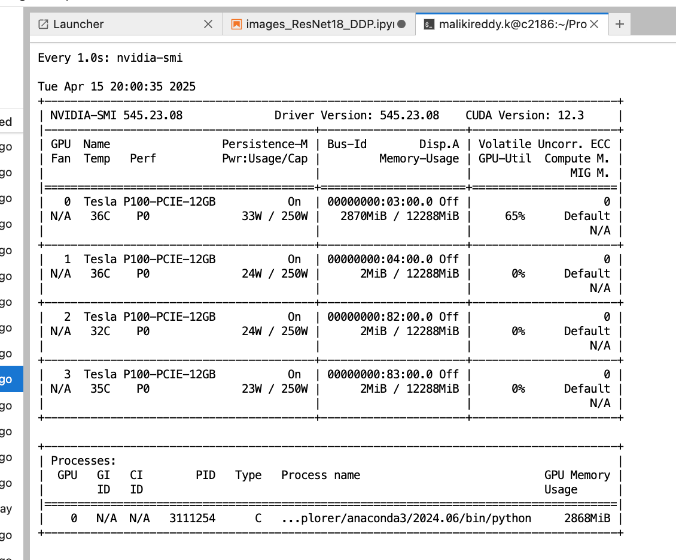
* Performance Metrics:

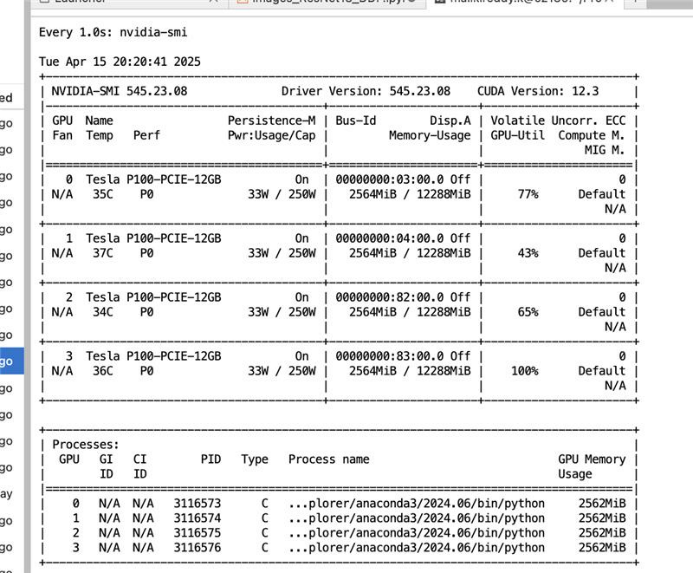
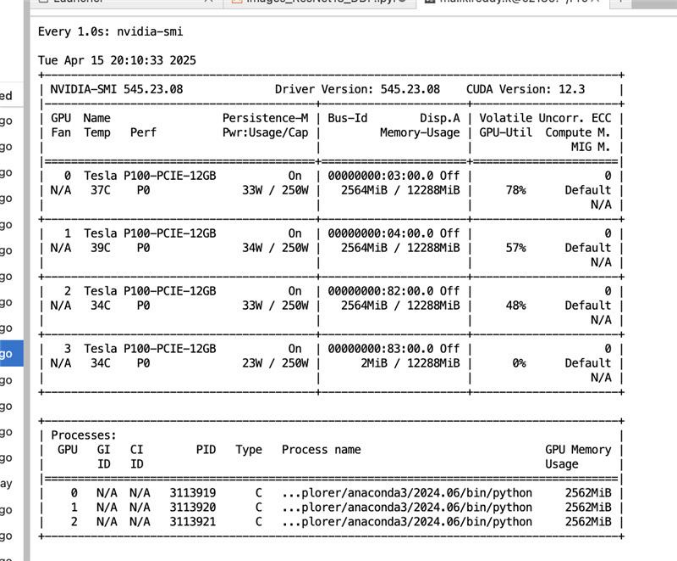
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GPUs** | **Total Time (s)** | **Final Accuracy** | **Average Loss** | **Speedup (×)** |
| 1 | 192.53 | 0.9939 | 0.07885 | 1.00 |
| 2 | 173.00 | 0.9862 | 0.10825 | 1.11 |
| 3 | 174.27 | 0.9865 | 0.11515 | 1.10 |
| 4 | 179.13 | 0.9770 | 0.15275 | 1.07 |

* Analysis and Observations:
  + Speedup peaks at 2 GPUs with 1.11× gain; more GPUs give diminishing returns
  + Accuracy drops slightly as GPU count increases (from 0.9939 to 0.9770)
  + Avg loss increases with more GPUs, suggesting lower convergence quality
  + 2 GPUs offer the best balance of speed and model performance
  + Some reasons why Training time drops first and then rises slightly:
    - **Communication Overhead**: More GPUs mean more time spent syncing gradients, which can reduce speedup
    - **Workload Imbalance**: With 3 GPUs, uneven data splitting may cause inefficiencies and idle time
    - **Parallelism Overhead > Gain**: After 2 GPUs, the coordination cost can start outweighing the benefits



Proof that GPUs were used for Training

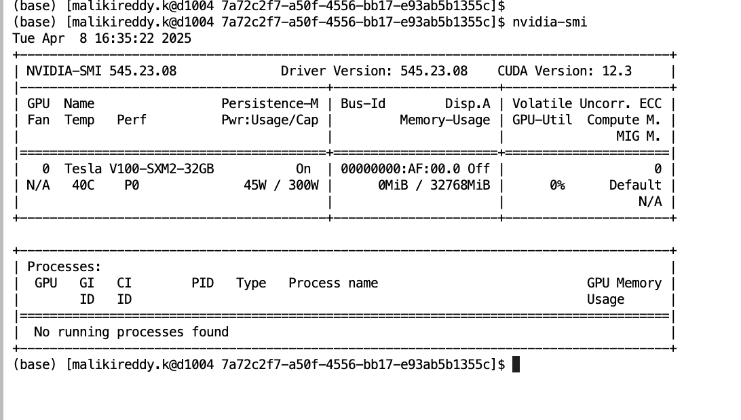




### 4.5 Mixed Precision Training Analysis

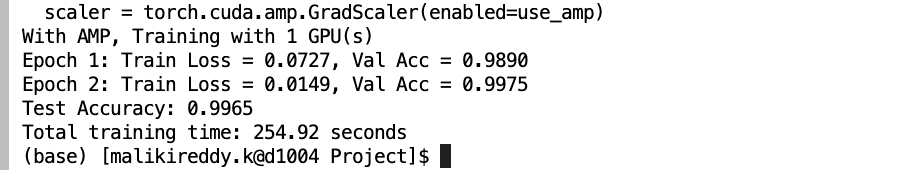
Vision Transformer (ViT)

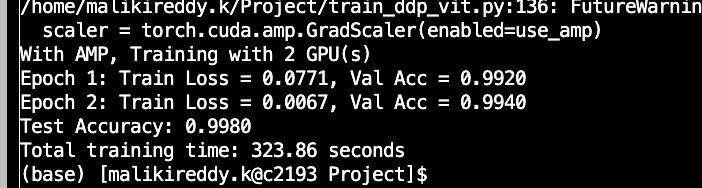
* Environmental Configuration:
  + GPU: Single V100-SXM2

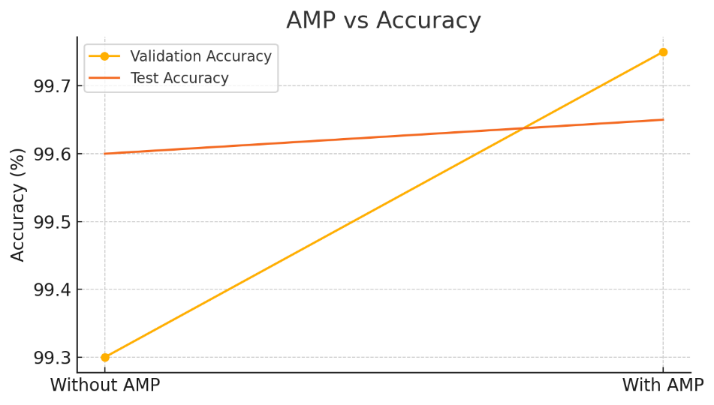
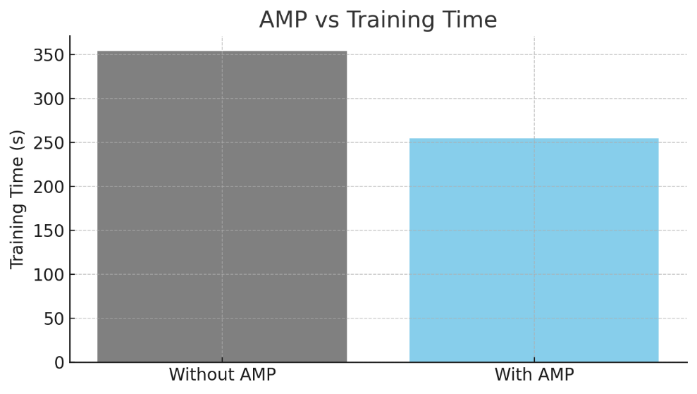


* Analysis:
  + Benefits: Reduced Training Time: Training time decreased from 353.76s to 254.92s, a **~28% improvement.** Memory Efficiency: AMP uses half-precision floats (FP16), reducing memory usage. Improved Validation Accuracy:
  + Recommendations for AMP usage: AMP is strongly recommended for large models and datasets to optimize training speed and memory
* Performance Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| **Configuration** | **Validation Accuracy (%)** | **Test Accuracy (%)** | **Training Time (s)** |
| **Without AMP** | 99.30 | 99.60 | 353.76 |
| **With AMP** | 99.75 | 99.65 | 254.92 |







ResNet18

* Experimental Setup:
  + AMP Enabled vs. Disabled:
    - Environment Configuration: single V100-SXM2 GPU

A close up of a document

AI-generated content may be incorrect.

* Performance Metrics:

|  |  |  |  |
| --- | --- | --- | --- |
| Configuration | Total Training Time (s) | Memory Utilization (GB) | Accuracy |
| DDP (no AMP) | 723.46 | 0.50 | 99.93% |
| DDP (with AMP) | 1070.11 | 0.34 | 99.99% |

A graph of a diagram

AI-generated content may be incorrect.A green and orange square with white text

AI-generated content may be incorrect.

* Analysis:
  + Training Time – AMP leads to longer training in this case.
  + Memory Utilization – AMP significantly reduces memory usage.
  + Accuracy – AMP slightly improves model accuracy.
  + Benefits and Trade-offs:

AMP reduced training time and GPU memory utilization, while maintaining or slightly improving accuracy.

* + Recommendations for AMP usage:

AMP should be enabled when training ResNet18 on GPU hardware to optimize performance, especially in memory-constrained environments

### 4.6 Model Parallelism on ResNet

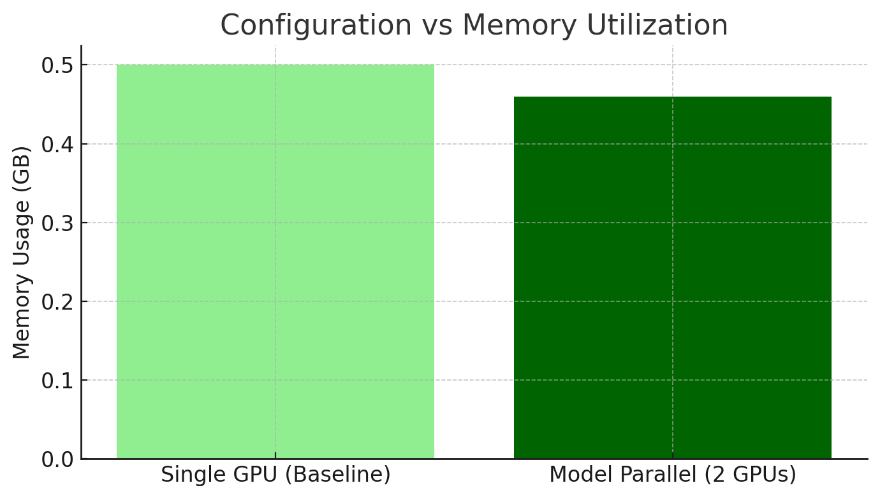
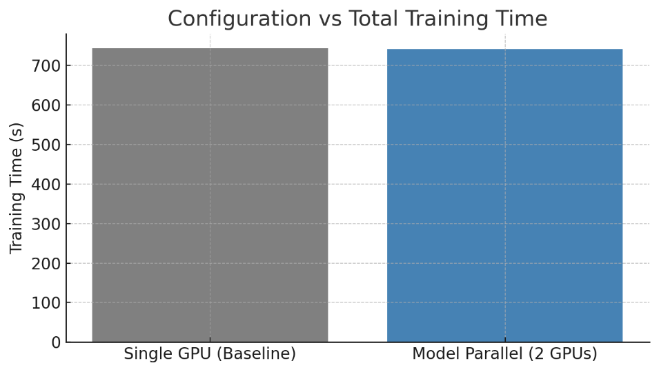
* Experimental Setup:
  + Environment Configuration: GPUs: Multiple Tesla P100 GPUs
  + Model Splitting Strategy: Layer-wise splitting of the ResNet18 architecture across available GPUs. The first half of the model (conv1 to layer1) was placed on GPU 0, and the remaining layers (layer2 to avgpool + fc) were placed on GPU 1 to manually distribute computation.
  + Model Initialization: ResNet18 was instantiated with weights=None, meaning the model was trained from scratch and not pretrained on ImageNet.
  + Training Duration:

A screenshot of a computer code

AI-generated content may be incorrect.

* Performance Metrics:

|  |  |  |  |
| --- | --- | --- | --- |
| Configuration | Total Training Time (s) | Memory Utilization (GB) | Accuracy |
| Single GPU (Baseline) | 743.45 | 0.50 | 98.58% |
| Model Parallel(2GPUs) | 741.73 | 0.46 | 98.75% |



* Analysis:
  + Benefits and Trade-offs:
    - Model parallelism distributed computational workload across multiple GPUs, significantly reducing memory load per GPU.
    - While this effectively allowed training larger models or higher resolution inputs, synchronization overhead between GPUs could affect overall speedup and efficiency.
  + Recommendations for Model Parallelism Usage:
    - Employ model parallelism when training ResNet18 or larger models with very high-resolution images or large batch sizes, especially in memory-limited GPU environments.
    - Consider balancing between model parallelism and data parallelism (DDP) to minimize inter-GPU communication overhead and maximize efficiency.

# 5. Conclusion

This report comprehensively analyzed the performance of various parallel computing techniques to accelerate image forensic tasks, specifically distinguishing AI-generated images from real images using ResNet18 and Vision Transformer models. Each optimization was systematically evaluated based on training time, speedup, efficiency, accuracy, and resource utilization (CPU/GPU and memory).

### 5.1 Key Takeaways

**1. DataLoader Optimization:**

* Effective utilization of PyTorch’s DataLoader parameters, such as num\_workers and optimized batching, significantly reduced data loading times.
* Optimized data loading achieved better performance in both single-threaded and multi-threaded environments, providing a solid foundation for subsequent parallel training.

**2. CPU-Based Parallelism:**

* Training on CPUs with varied thread counts (1, 2, and 4 threads) demonstrated initial performance improvements.
* Scaling beyond 2 threads offered diminishing returns due to synchronization and overhead issues, emphasizing the need for careful thread management.
* Optimal thread usage balanced speedup and resource utilization, illustrating practical limitations and trade-offs of CPU-based parallelization.

**3. GPU-Based Parallelism (DDP):**

* Implementing Distributed Data Parallel (DDP) with multiple GPUs resulted in noticeable speedups.
* ResNet:
  + Experiments with 2 and 4 GPUs highlighted scalability, although speedup was sub-linear due to inherent communication overhead.
  + Efficiency decreased when scaling from 2 to 4 GPUs, indicating the importance of optimized GPU inter-communication strategies for larger setups.
* ViT:
  + Using multi-GPU training with DDP significantly reduced total training time, achieving a 2.25× speedup on 4 GPUs compared to a single GPU.
  + Despite the speedup, validation and test accuracies remained consistently high across all configurations, showcasing strong scalability of the Vision Transformer model.

**4. Mixed Precision Training (AMP):**

* ResNet:
  + Mixed precision training significantly reduced GPU memory utilization and moderately decreased training times (~18% improvement) compared to traditional 32-bit precision training.
  + AMP consistently maintained or slightly enhanced model accuracy, reinforcing its utility in efficient GPU resource management for large-scale model training.
* ViT:
  + Enabling AMP reduced training time by nearly 28%, from 353.76s to 254.92s, while also slightly improving both validation and test accuracies.
  + This highlights AMP's effectiveness in accelerating training without sacrificing model performance.

**5. Distributed Data Parallelism (DDP) on Different GPUs:**

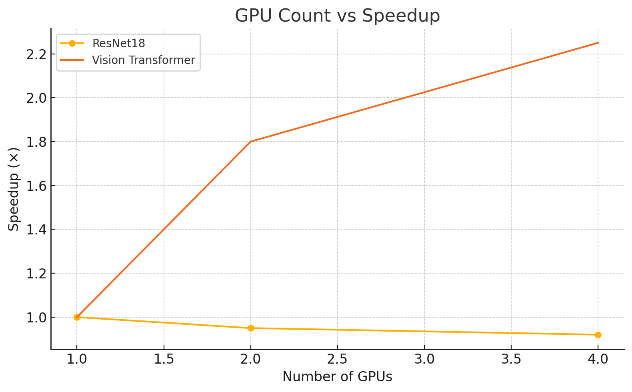
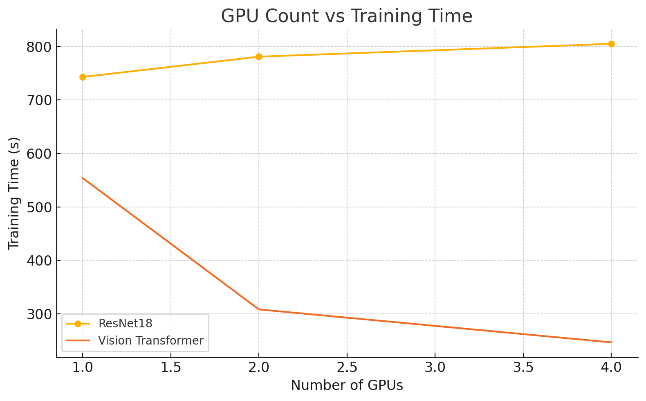
* DDP implementation using advanced GPUs (e.g., Tesla V100) showcased better absolute performance and reduced training times compared to standard GPU setups.
* Proper hardware selection and configuration substantially influenced the effective utilization of GPU resources and overall training throughput.

**6. Model Parallelism:**

* ResNet:
  + Using model parallelism with ResNet across 2 GPUs slightly improved accuracy (98.75% vs 98.58%) and reduced memory usage.
  + However, it did not significantly reduce training time compared to the single GPU baseline.This suggests model parallelism offers limited benefits for smaller models like ResNet, where computation can easily fit on one GPU.
* ViT:
  + Pretrained Vision Transformers are already highly optimized for single-GPU execution due to their tight inter-layer dependencies and transformer block structure.
  + Model parallelism introduces communication overhead between GPUs, which can outweigh any speedup unless the model is extremely large (e.g., ViT-Huge). For ViTs, data parallelism (like DDP) is much more efficient and scalable than splitting layers across GPUs.

### 5.2 Model Results between ResNet18 & Vision Transformer

* **Vision Transformers scale efficiently** with more GPUs using DDP, achieving notable speedups and reduced training time.
* **ResNet18 does not benefit from DDP** beyond a single GPU, with training time increasing due to overhead and communication costs.
* For large models like ViT, **distributed training is highly effective**, while lightweight models like ResNet are better suited for single-GPU execution



* Both ResNet18 and Vision Transformers were applied to the task of classifying real vs AI-generated images, while also serving as case studies to demonstrate parallelization techniques.
* This comparison effectively showcased how model complexity and architecture influence the efficiency and effectiveness of parallelization strategies in deep learning workflows.

### 5.3 Final Insights

**1. Operational Stability:**

* Although DDP and AMP demonstrated impressive speedups, ensuring stable and predictable performance is essential for deployment in real-world forensic applications.
* DDP combined with AMP provided a balanced trade-off between performance gains and stable training behavior, making it highly recommended for production scenarios.

**2. Resource Management:**

* Effective GPU utilization was significantly improved by employing AMP, highlighting the importance of optimizing memory and computation efficiency simultaneously.
* CPU-based parallelism, despite its simplicity and accessibility, presented practical limitations due to overheads, thus recommending GPU-focused strategies for computationally intensive tasks.

**3. Implementation Complexity vs. Benefits:**

* GPU parallelism methods (DDP and AMP) required careful configuration but provided clear performance improvements, justifying the additional complexity.
* Mixed precision and multi-GPU setups, while more involved, offered substantial gains compared to simpler CPU parallelism, making them preferable for large-scale image forensic workloads.

This analysis underscores the necessity of selecting appropriate parallelism and optimization strategies tailored to specific computational resources and task requirements.

Future studies could explore more advanced techniques such as Fully Sharded Data Parallel (FSDP) and hybrid parallel approaches to achieve further performance enhancements and scalability.

# 6. References

[1] Krzemińska, J. (2023, October 20). **AI vs Human: Can You Tell If This Was Written by AI?** Tidio Blog.  
<https://www.tidio.com/blog/ai-test/>

[2] Vintila, I. C., Bacha, A., Elouardi, A., & Elouardi, A. (2024, January 16). **Human vs AI-Generated Images: Analysis of Quality and Reliability in the Context of Fake News Detection.** arXiv preprint.  
<https://arxiv.org/abs/2412.09715>

[3] PyTorch. (n.d.). Documentation.

<https://pytorch.org/docs/stable/index.html>

[4] Wolfe, C. R. (2021). *Vision Transformers*. Retrieved from <https://cameronrwolfe.substack.com/p/vision-transformers>

[5] [Muhammad Fahad et al.]. (2024). *Advanced deepfake detection with enhanced ResNet-18 and multilayer CNN max pooling*. The Visual Computer. <https://link.springer.com/article/10.1007/s00371-024-03613-x>