

Project Report
On
Distributed AI Model Training using MPI and GPU Acceleration



Submitted
In partial fulfilment
For the award of the Degree of

Advance Certificate Course – High Performance Computing
(C-DAC, ACTS (Pune))

Guided By:

Mr. Sanjay Sane Sir

Submitted By:

Name	PRN
Aishwarya Wankhade	250840150003
Nikita Nasare	250840150021
Prajwal Choudhari	250840150026
Shruti Martode	250840150036
Srushti Patil	250840150040

Centre for Development of Advanced Computing
(C-DAC), ACTS (Pune- 411008)

ACKNOWLEDGEMENT

This is to acknowledge our indebtedness to our Project Guide, **Mr. Sanjay Sane Sir** C-DAC ACTS, Pune for his constant guidance and helpful suggestion for preparing this project **Distributed AI Model Training using MPI and GPU Acceleration**. We express our deep gratitude towards him for inspiration, personal involvement, constructive criticism that he provided us along with technical guidance during the course of this project.

We take this opportunity to thank Head of the department **Mr. Gaur Sunder** for providing us such a great infrastructure and environment for our overall development.

We express sincere thanks to **Mr. Saumyakant Behra**, Process Owner, for their kind cooperation and extendible support towards the completion of our project.

It is our great pleasure in expressing sincere and deep gratitude towards **Mrs. Risha P R (Program Head)** and **Ms. Pratiksha Gacche** (Course Coordinator, PG-DBDA) for their valuable guidance and constant support throughout this work and help to pursue additional studies.

Also, our warm thanks to **C-DAC ACTS Pune**, which provided us this opportunity to carry out, this prestigious Project and enhance our learning in various technical fields.

Aishwarya Wankhede	250840150003
Nikita Nasare	250840150021
Prajwal Choudhari	250840150026
Shruti Martode	250840150036
Srushti Patil	250840150040

ABSTRACT

The growing computational demands of deep learning models make parallel and distributed training essential for efficient and scalable learning. This project explores Distributed AI Model Training using GPU acceleration within a high-performance computing (HPC) environment for image classification with the MobileNetV2 architecture trained on the CIFAR-10 dataset. Input images are resized to 224×224 and enhanced using data augmentation techniques.

Two training configurations are evaluated: serial training on a single GPU and distributed training across multiple GPUs using MPI and PyTorch Distributed Data Parallel (DDP) on HPC infrastructure. The distributed approach partitions data across processes and synchronizes model parameters through gradient communication. Performance is measured using training time, speedup, and classification accuracy.

Results demonstrate that HPC-enabled distributed GPU training significantly reduces training time compared to serial execution while maintaining comparable accuracy, confirming the effectiveness of HPC-based distributed deep learning frameworks for scalable AI model training.

Keywords: CIFAR-10, MobileNetV2, Distributed Training, MPI, PyTorch, Gradio, Image Classification, Deep Learning, HPC

Table of Contents

S. No	Title	Page No.
1	Introduction	01-02
1.1	Introduction	01
2	Objective and Specifications	03
3	Literature Review	04
4	Methodology and Techniques	05-09
4.1	Environment Setup	05
4.2	Dataset Preparation	08
4.3	Model Architecture and Model Architecture Diagram	06
4.4	Inference and Deployment	08
5	Implementation Details	09-13
5.1	Use of Python Platform for Writing the Code	10
5.2	Hardware and Software Configuration	10
5.3	Serial Implementation	11-14
5.4	Distributed Implementation (Multi-GPU/Multi-Node)	15-20
5.5	Inference Interface using Gradio	21-22
6	Results	23-27
6.1	Training CPU, GPU, MPI+DDP	23-25
6.2	Performance & Accuracy Analysis Table (CIFAR-10)	26-27
7	Conclusion	28
8	Future Enhancements	29
9	References	30

I. INTRODUCTION

Deep learning has become a core technology for solving complex problems in computer vision, speech recognition, and natural language processing. However, as neural network models grow in depth and complexity, the computational resources required for training increase significantly. Training deep learning models on large datasets using a single processing unit can be time-consuming and inefficient. To address these challenges, parallel and distributed training techniques have emerged as effective solutions for accelerating model training and improving scalability.

Image classification is a fundamental task in computer vision, widely used in applications such as object recognition, autonomous systems, and medical imaging. The CIFAR-10 dataset is a standard benchmark for evaluating image classification models, consisting of 60,000 images across 10 categories. Although CIFAR-10 images are small in size, modern convolutional neural networks benefit from higher input resolutions and advanced augmentation strategies to improve feature learning and generalization.

This project focuses on Distributed AI Model Training for Parallel Machine Learning using the MobileNetV2 architecture, a lightweight and efficient convolutional neural network designed for high performance with reduced computational cost. The model is trained using GPU acceleration and evaluated under two configurations: serial training on a single GPU and distributed training across multiple GPUs using Message Passing Interface (MPI) and PyTorch Distributed Data Parallel (DDP). In the distributed setup, the dataset is partitioned across multiple processes, and model parameters are synchronized through gradient communication to ensure consistent learning.

The project aims to analyze the performance benefits and limitations of distributed training by comparing training time, scalability, and classification accuracy across different execution modes. By implementing and evaluating both serial and distributed approaches, this work provides practical insights into the effectiveness of distributed deep learning frameworks and demonstrates how parallel computing can be leveraged to build scalable and efficient AI systems.

Key Contributions of This Project:

- Implementation of MobileNetV2 on CIFAR-10 with 224×224 input resolution (upscaled from 32×32)
- Strong data augmentation pipeline including random resized cropping, color jittering, and rotation
- Serial training implementation optimized for single GPU environments
- Distributed training implementation using MPI and DDP for multi-GPU/multi-node scenarios
- Web-based inference interface using Gradio for production deployment
- Comprehensive comparative analysis of training efficiency and scalability

II. Objective

1. Develop an efficient CIFAR-10 image classification system using MobileNetV2 architecture.
2. Implement single-GPU and CPU serial training for baseline performance measurement.
3. Develop multi-GPU distributed training using MPI and PyTorch DDP.
4. Create a user-friendly web interface for model inference and deployment.
5. Evaluate scalability and performance improvements with distributed training.
6. Demonstrate best practices in distributed machine learning on HPC systems.

III. LITERATURE REVIEW

MobileNetV2 Architecture: Howard et al. (2017) introduced MobileNetV2, which employs inverted residual blocks instead of traditional residual blocks[1]. The key innovation is the linear bottleneck layer that reduces spatial dimensions without using ReLU activation, preventing information loss during feature compression. This architecture achieves state-of-the-art accuracy-to-parameter ratio on ImageNet and downstream tasks.

Distributed Training Approaches: Goyal et al. (2017) demonstrated that synchronized data parallelism with gradient averaging achieves near-linear scaling on multiple GPUs[2]. Lin et al. (2019) introduced gradient compression techniques that further improve communication efficiency in distributed settings[3].

Data Augmentation Strategies: Cubuk et al. (2018) showed that stronger data augmentation techniques, including random resized cropping and color jittering, substantially improve generalization on image classification tasks[4]. Zhong et al. (2020) demonstrated that appropriate augmentation choices significantly impact model performance, especially for smaller datasets like CIFAR-10[5].

MPI in Deep Learning: While traditional deep learning frameworks often use PyTorch's native distributed capabilities, integration with MPI provides additional flexibility for HPC environments and enables seamless integration with existing MPI-based workflows[6].

Transfer Learning: Fine-tuning pre-trained ImageNet models on smaller datasets like CIFAR-10 has been shown to provide significant advantages over training from scratch, reducing training time and achieving better accuracy due to learned low-level features[7].

Web-Based ML Deployment: Gradio, developed by Hugging Face, provides a lightweight framework for creating shareable web interfaces for machine learning models[8]. It eliminates deployment complexity and makes models accessible to non-technical stakeholders.

IV . METHODOLOGY

Methodology and Techniques

4.1 Environment Setup

- **Frameworks:** PyTorch for model development and training, torchvision for datasets and pre-trained models, PIL for image processing, mpi4py for MPI communication, Gradio for web deployment.
- **Hardware:** GPU-enabled system for acceleration; CPU fallback supported. For distributed training, multiple GPUs or multiple nodes in an HPC cluster.
- **Device Selection:** Automatically detects GPU using `torch.device("cuda" if torch.cuda.is_available() else "cpu")`. In distributed mode, each process is assigned a unique GPU.
- **Reproducibility:** Fixed random seed (`torch.manual_seed(42)`) ensures consistent results across runs.

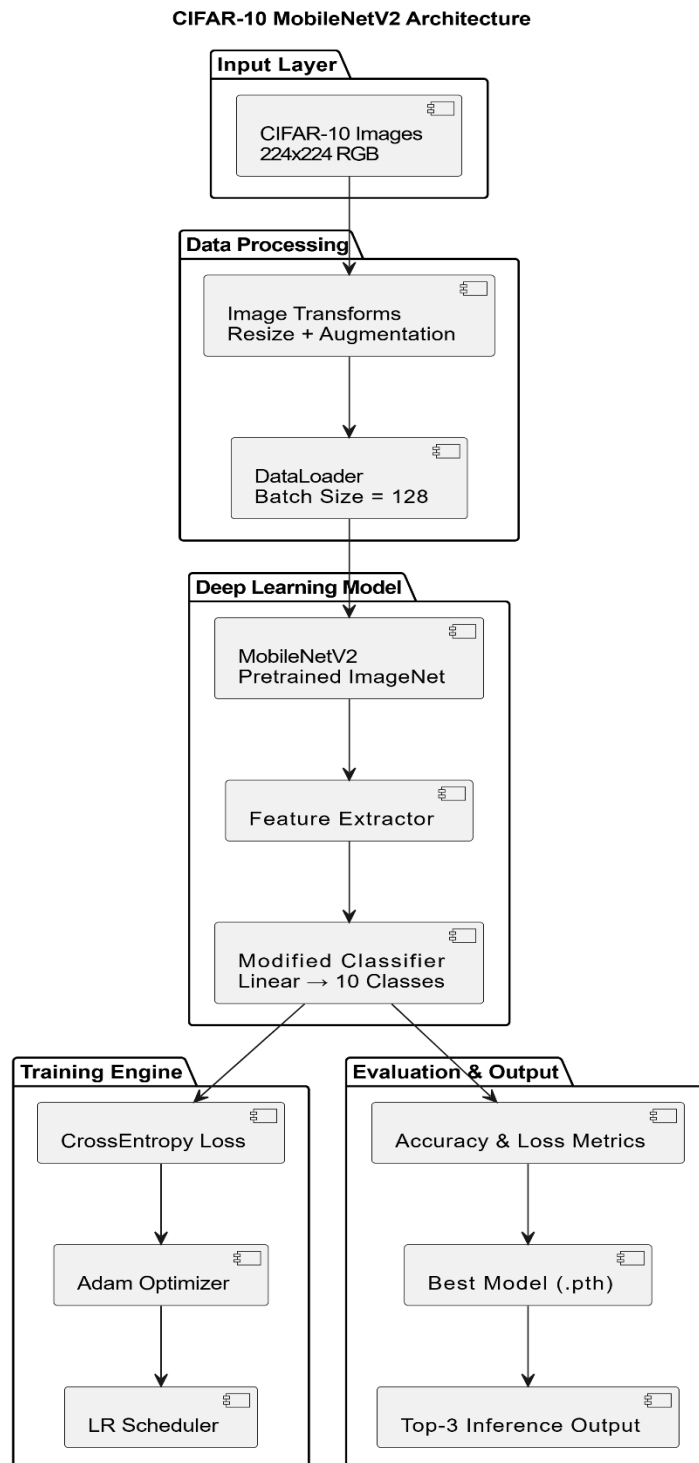
4.2 Dataset Preparation

- **Dataset:** CIFAR-10, consisting of 60,000 32×32 color images in 10 classes (50,000 training, 10,000 testing).
- **Image Transformations:**
 - **Training (Serial + Distributed):** Resize to 224×224 → RandomResizedCrop → Horizontal Flip → ColorJitter → RandomRotation → ToTensor → Normalize (ImageNet mean & std).
 - **Testing/Inference:** Resize → ToTensor → Normalize (no augmentation).
- **Distributed Sampling:** For multi-GPU training, DistributedSampler ensures each process works on a unique subset of data, preventing overlap and ensuring efficiency.
- **Batch Sizes:** 64 for serial training, 128 per process for distributed training.

4.3 Model Architecture

- **Base Model:** MobileNetV2, chosen for lightweight computation, efficiency, and strong accuracy.
- **Modification for CIFAR-10:** Replace classifier layer with `nn.Linear(in_features, 10)` to match 10 classes.
- **Deployment:** Model moved to GPU if available; in DDP, wrapped in `torch.nn.parallel.DistributedDataParallel` for synchronized gradient updates.
- **Pretraining:** Use ImageNet weights to accelerate convergence and improve generalization.

Model Architecture Diagram



Training Procedures

Serial Single-GPU Training

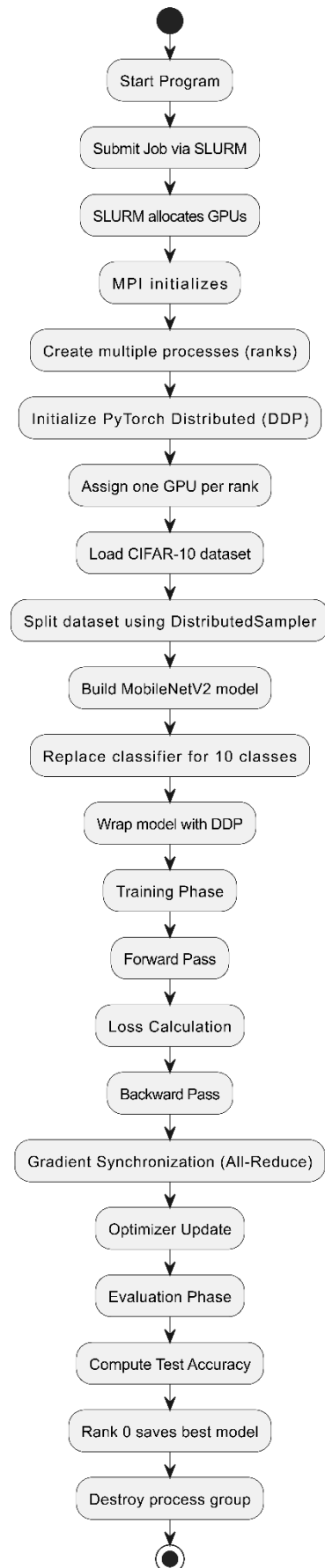
- **Purpose:** Establish baseline performance metrics (accuracy, loss, training time).
- **Loss Function:** Cross-entropy.
- **Optimizer:** Adam with learning rate 1e-4.
- **Scheduler:** StepLR reduces learning rate every 10 epochs by factor 0.1.
- **Loop:**
 1. Forward pass → compute loss → backward pass → optimizer step.
 2. Track training loss and accuracy.
 3. Evaluate on test set after each epoch.
- **Checkpointing:** Save best model based on test accuracy (best_mobilenetv2_cifar10_224.pth).

Distributed Multi-GPU Training (MPI + DDP)

- **Purpose:** Reduce training time, scale across multiple GPUs/nodes.
- **Distributed Initialization:**
 - MPI manages processes across GPUs/nodes.
 - PyTorch DDP synchronizes gradients across processes.
 - Backend selection: nccl for GPU, gloo for CPU.
- **Training Loop:**
 - Each process receives a subset of training data via DistributedSampler.
 - Forward pass → loss computation → backward pass → optimizer step.
 - Aggregate training and test losses across ranks using dist.all_reduce.
 - Scheduler updates learning rate per epoch.
- **Checkpointing:** Save best-performing model (best_mobilenetv2_cifar10_224_mpi.pth).

Flow Diagram of Distributed (MPI+DDP) code:

Flow Diagram: MPI + DDP CIFAR-10 Training



4.4 Inference and Deployment

- **Serial Inference:**
 - Input image preprocessed with test transforms.
 - Forward pass → softmax → top-k predictions with confidence.
 - Confidence thresholding flags uncertain predictions.
- **Web Deployment:** Gradio interface enables users to upload images and receive top-3 predictions interactively.

Performance Monitoring and Evaluation

- **Metrics:**
 - Training loss, validation loss, accuracy.
 - Speedup comparisons between serial and distributed training.
 - Resource utilization (GPU memory, computation time).
- **Scalability Analysis:** Distributed training reduces total training time; efficiency evaluated by comparing single vs. multi-GPU runtimes.

Techniques and Best Practices

- **Data Augmentation:** Improves generalization and robustness.
- **Transfer Learning:** Pretrained weights accelerate convergence.
- **Device-Aware Computation:** Efficient use of GPU resources; avoids overloading any device.
- **Gradient Synchronization:** DDP ensures consistent model updates across processes.
- **Reproducibility:** Fixed seeds, consistent transforms, and checkpointing.
- **Deployment Readiness:** Gradio interface enables easy testing and inference.

V. IMPLEMENTATION DETAILS

5.1 Use of Python Platform for Writing the Code

The project is implemented entirely using **Python**, leveraging the following libraries and frameworks:

- **PyTorch:** Core deep learning framework used for creating, training, and evaluating the MobileNetV2 model.
- **torchvision:** Provides CIFAR-10 dataset utilities, pre-trained models, and image transformations for data augmentation and preprocessing.
- **MPI for Python (mpi4py):** Enables distributed multi-GPU/multi-node training on HPC clusters.
- **Gradio:** Used to create a user-friendly web interface for model inference.
- **PIL (Python Imaging Library):** Handles image loading and preprocessing.
- **NumPy:** For numerical computations and array operations.

5.2 Hardware and Software Configuration

Hardware Configuration:

- **Supercomputer:** PARAM Utkarsh / Param HPC system
- **CPU:** 2× Intel Xeon Gold 6248 CPUs @ 2.50GHz, 40 cores total (20 cores per socket)
- **GPU:** 2× NVIDIA Tesla V100-SXM2, 16 GB memory each
- **RAM:** High memory nodes ≥ 64 GB (as allocated per job)
- **NUMA Nodes:** 2

Software Configuration:

- **Python 3.x:** Programming language for model development, training, and deployment
- **PyTorch:** Deep learning framework for serial and distributed training

- **torchvision:** Provides CIFAR-10 dataset, pre-trained models, and image augmentation utilities
- **MPI / OpenMPI:** For running distributed training across multiple nodes
- **CUDA Toolkit & cuDNN:** GPU acceleration (CUDA 11.0, NVIDIA driver 450.51.05)
- **Anaconda:** Python environment management and package installation

5.3 Serial Implementation

Slurm Script:

```
#!/bin/bash
#SBATCH --job-name=cifar10_mobilenet_cpu
#SBATCH --output=logs_cifar10_cpu_%j.out
#SBATCH --error=logs_cifar10_cpu_%j.err
#SBATCH --time=15:00:00      # adjust as needed
#SBATCH --partition=cpu      # your CPU partition name
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4    # number of CPU cores
#SBATCH --mem=16G            # RAM

# Load modules / activate env (adapt to your cluster)
module load python/3.10
module load anaconda3/anaconda3

conda activate py395_env
/home/chuk372/cpu_serial

python cifar10_serial_mobilenet_224.py
```

Code: CPU SERIAL

```
# cifar10_serial_mobilenet_224.py
# CIFAR-10 training with MobileNetV2 (stronger augmentations, 224x224, top-3 inference)

import time
from copy import deepcopy

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
```

```
import torchvision
import torchvision.transforms as transforms
from torchvision import models
from PIL import Image
import torch.nn.functional as F

# ----- Device -----

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)

# CIFAR-10 classes
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck']

# ----- Transforms -----

IMG_SIZE = 224 # increased from 128 for more detail

train_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomResizedCrop(IMG_SIZE, scale=(0.7, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.3, contrast=0.3,
                           saturation=0.3, hue=0.1),
    transforms.RandomRotation(degrees=15),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])

test_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])

# ----- CIFAR-10 data -----

print("Downloading CIFAR-10 (if needed)...")
train_dataset = torchvision.datasets.CIFAR10(
    root="./data", train=True, download=True, transform=train_transform
)
test_dataset = torchvision.datasets.CIFAR10(
    root="./data", train=False, download=True, transform=test_transform
)
```

```

batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size,
                           shuffle=True, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
                          shuffle=False, num_workers=2)

print("Train samples:", len(train_dataset))
print("Test samples:", len(test_dataset))

# ----- MobileNetV2 model -----

model = models.mobilenet_v2(pretrained=True)
in_features = model.classifier[1].in_features
model.classifier[1] = nn.Linear(in_features, 10)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

total_params = sum(p.numel() for p in model.parameters())
print("Total parameters:", total_params)
print("Starting serial training...\n")

# ----- Training loop -----

EPOCHS = 20
best_acc = 0.0
best_state = deepcopy(model.state_dict())
total_start = time.time()

for epoch in range(EPOCHS):
    epoch_start = time.time()

    # ---- train ----
    model.train()
    running_loss = 0.0
    running_corrects = 0

    for imgs, labels in train_loader:
        imgs, labels = imgs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(imgs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

```

```

_, preds = torch.max(outputs, 1)
running_loss += loss.item() * imgs.size(0)
running_corrects += torch.sum(preds == labels).item()

epoch_loss = running_loss / len(train_dataset)
epoch_acc = running_corrects / len(train_dataset)

# ---- evaluate ----
model.eval()
test_loss_sum = 0.0
test_correct = 0

with torch.no_grad():
    for imgs, labels in test_loader:
        imgs, labels = imgs.to(device), labels.to(device)
        outputs = model(imgs)
        loss = criterion(outputs, labels)
        _, preds = torch.max(outputs, 1)
        test_loss_sum += loss.item() * imgs.size(0)
        test_correct += torch.sum(preds == labels).item()

test_loss = test_loss_sum / len(test_dataset)
test_acc = test_correct / len(test_dataset)

scheduler.step()
epoch_time = time.time() - epoch_start

print(
    f"Epoch {epoch+1}/{EPOCHS} "
    f"Time: {epoch_time:.2f}s "
    f"Train Loss: {epoch_loss:.4f} "
    f"Train Acc: {epoch_acc:.4f} "
    f"Test Loss: {test_loss:.4f} "
    f"Test Acc: {test_acc:.4f}"
)

if test_acc > best_acc:
    best_acc = test_acc
    best_state = deepcopy(model.state_dict())

total_time = time.time() - total_start
print(f"\nBest test accuracy: {best_acc:.4f}")
print(f"Total training time: {total_time:.2f}s ({total_time/60:.2f} min)")

model.load_state_dict(best_state)
torch.save(model.state_dict(), "best_mobilenetv2_cifar10_224.pth")
print("Saved best_mobilenetv2_cifar10_224.pth")

```

5.4 Distributed Implementation

Slurm Script

```
#!/bin/bash
#SBATCH --job-name=cifar_mpi_gpu
#SBATCH --partition=gpu          # GPU partition
#SBATCH --nodes=1
#SBATCH --ntasks=2              # 1 MPI rank per GPU
#SBATCH --gres=gpu:2            # each gpu node has 2 GPUs
#SBATCH --time=12:00:00
#SBATCH --output=cifar_mpi_gpu128_%j.out
#SBATCH --error=cifar_mpi_gpu128_%j.err

module purge
module load anaconda3/anaconda3
module load openmpi/4.1.1
source /home/apps/anaconda3/etc/profile.d/conda.sh
conda activate py395_env        # env with CUDA-capable torch + mpi4py

cd "$HOME/cifar_10" || exit 1

mpirun --mca pm1 ob1 --mca btl tcp,self -np 2 \
  python cifar10_mpi_mobilenet_224.py
```

Code: GPU(MPI+DDP)(2 GPU)

```
# cifar10_mpi_mobilenet_224.py
# CIFAR-10 + MobileNetV2 + MPI + DDP, 224x224, same augments as serial version

import os
import time
from copy import deepcopy

import torch
import torch.nn as nn
import torch.optim as optim
import torch.distributed as dist
import torch.nn.functional as F

from mpi4py import MPI
from torch.utils.data import DataLoader
from torch.utils.data.distributed import DistributedSampler

import torchvision
import torchvision.transforms as transforms
from torchvision import models

# ----- MPI + Distributed init -----
```

```

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
world_size = comm.Get_size()

def setup_distributed():
    os.environ["MASTER_ADDR"] = "localhost"
    os.environ["MASTER_PORT"] = "29500"
    os.environ["RANK"] = str(rank)
    os.environ["WORLD_SIZE"] = str(world_size)

    backend = "nccl" if torch.cuda.is_available() else "gloo"
    dist.init_process_group(backend=backend, rank=rank, world_size=world_size)

    if torch.cuda.is_available():
        local_rank = rank % torch.cuda.device_count()
        torch.cuda.set_device(local_rank)
        device = torch.device(f"cuda:{local_rank}")
    else:
        local_rank = 0
        device = torch.device("cpu")

    return device, local_rank, backend

def cleanup():
    dist.destroy_process_group()

# ----- Main -----

def main():
    if rank == 0:
        print("=" * 70)
        print("MPI + DDP MobileNetV2 CIFAR-10 (224x224)")
        print("=" * 70)

    torch.manual_seed(42)
    device, local_rank, backend = setup_distributed()

    if rank == 0:
        print(f"Backend: {backend}, World size: {world_size}, Device(rank0): {device}")

    # CIFAR-10 classes
    classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

    # ----- transforms: 224x224 + strong augments -----

```

```

IMG_SIZE = 224

train_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomResizedCrop(IMG_SIZE, scale=(0.7, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.3, contrast=0.3,
                           saturation=0.3, hue=0.1),
    transforms.RandomRotation(degrees=15),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])

test_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])

# ----- CIFAR-10 datasets (rank 0 downloads) -----

if rank == 0:
    print("Downloading CIFAR-10 (if not present)...")
    _ = torchvision.datasets.CIFAR10(
        root="./data", train=True, download=True,
transform=train_transform
    )
    _ = torchvision.datasets.CIFAR10(
        root="./data", train=False, download=True,
transform=test_transform
    )

    dist.barrier()

train_dataset = torchvision.datasets.CIFAR10(
    root="./data", train=True, download=False, transform=train_transform
)
test_dataset = torchvision.datasets.CIFAR10(
    root="./data", train=False, download=False, transform=test_transform
)

```

```

if rank == 0:
    print("Train samples:", len(train_dataset))
    print("Test samples:", len(test_dataset))

# ----- Distributed samplers + loaders -----

batch_size = 128

train_sampler = DistributedSampler(
    train_dataset, num_replicas=world_size, rank=rank, shuffle=True
)
test_sampler = DistributedSampler(
    test_dataset, num_replicas=world_size, rank=rank, shuffle=False
)

train_loader = DataLoader(
    train_dataset, batch_size=batch_size,
    sampler=train_sampler, num_workers=2
)
test_loader = DataLoader(
    test_dataset, batch_size=batch_size,
    sampler=test_sampler, num_workers=2
)

# ----- MobileNetV2 model -----

model = models.mobilenet_v2(pretrained=True)
in_features = model.classifier[1].in_features
model.classifier[1] = nn.Linear(in_features, 10)
model = model.to(device)

ddp_model = torch.nn.parallel.DistributedDataParallel(
    model,
    device_ids=[local_rank] if torch.cuda.is_available() else None,
)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(ddp_model.parameters(), lr=1e-4)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

if rank == 0:
    total_params = sum(p.numel() for p in ddp_model.parameters())
    print(f"Total parameters: {total_params}")
    print("Starting distributed training...\n")

# ----- Training loop -----

```

```

EPOCHS = 20
best_acc = 0.0
best_state = deepcopy(ddp_model.module.state_dict())
total_start = time.time()

for epoch in range(EPOCHS):
    epoch_start = time.time()
    train_sampler.set_epoch(epoch)

    # ---- train ----
    ddp_model.train()
    running_loss = 0.0
    running_correct = 0
    running_total = 0

    for imgs, labels in train_loader:
        imgs, labels = imgs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = ddp_model(imgs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        _, preds = torch.max(outputs, 1)
        running_loss += loss.item() * imgs.size(0)
        running_correct += torch.sum(preds == labels).item()
        running_total += labels.size(0)

    # aggregate train loss across ranks
    train_loss_tensor = torch.tensor(
        [running_loss, running_total],
        device=device, dtype=torch.float64
    )
    dist.all_reduce(train_loss_tensor, op=dist.ReduceOp.SUM)
    global_loss = train_loss_tensor[0].item()
    global_total = train_loss_tensor[1].item()
    epoch_loss = global_loss / global_total
    epoch_acc = running_correct / running_total # local approx

    # ---- evaluate ----
    ddp_model.eval()
    test_correct = 0
    test_total = 0
    test_loss_sum = 0.0

    with torch.no_grad():
        for imgs, labels in test_loader:

```

```

        imgs, labels = imgs.to(device), labels.to(device)
        outputs = ddp_model(imgs)
        loss = criterion(outputs, labels)

        _, preds = torch.max(outputs, 1)
        test_loss_sum += loss.item() * imgs.size(0)
        test_correct += torch.sum(preds == labels).item()
        test_total += labels.size(0)

    # aggregate test loss across ranks
    test_loss_tensor = torch.tensor(
        [test_loss_sum, test_total],
        device=device, dtype=torch.float64
    )
    dist.all_reduce(test_loss_tensor, op=dist.ReduceOp.SUM)
    global_test_loss = test_loss_tensor[0].item()
    global_test_total = test_loss_tensor[1].item()
    test_loss = global_test_loss / global_test_total
    test_acc = test_correct / test_total # local approx

    scheduler.step()
    epoch_time = time.time() - epoch_start

    if rank == 0:
        print(
            f"Epoch {epoch+1}/{EPOCHS} "
            f"Time: {epoch_time:.2f}s "
            f"Train Loss: {epoch_loss:.4f} "
            f"Test Loss: {test_loss:.4f} "
            f"Test Acc(local): {test_acc:.4f}"
        )

    if test_acc > best_acc:
        best_acc = test_acc
        best_state = deepcopy(ddp_model.module.state_dict())

total_time = time.time() - total_start
if rank == 0:
    print(f"\nBest local test accuracy: {best_acc:.4f}")
    print(
        f"Total training time: {total_time:.2f}s "
        f"({total_time/60:.2f} min)"
    )
    torch.save(best_state, "best_mobilenetv2_cifar10_224_mpi.pth")
    print("Saved best_mobilenetv2_cifar10_224_mpi.pth")

cleanup()

```

```
if __name__ == "__main__":  
    main()
```

5.5 Inferencing with Gradio

```
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torchvision.transforms as transforms  
from torchvision import models  
from PIL import Image  
import gradio as gr  
  
# Device  
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  
  
# Rebuild model (same as training)  
model = models.mobilenet_v2(pretrained=False)  
model.classifier[1] = nn.Linear(model.last_channel, 10)  
model.to(device)  
  
# Load trained weights  
ckpt_path = "best_mobilenetv2_cifar10_224.pth"  
state = torch.load(ckpt_path, map_location=device)  
model.load_state_dict(state)  
model.eval()  
  
# CIFAR-10 class names  
classes = ["airplane", "automobile", "bird", "cat", "deer",  
           "dog", "frog", "horse", "ship", "truck"]  
  
# Same test transforms as training  
transform = transforms.Compose([  
    transforms.Resize((224, 224)),  
    transforms.ToTensor(),  
    transforms.Normalize((0.4914, 0.4822, 0.4465),  
                          (0.2023, 0.1994, 0.2010)),  
)  
  
def predict(img: Image.Image):  
    img = img.convert("RGB")  
    x = transform(img).unsqueeze(0).to(device)
```

```
with torch.no_grad():
    logits = model(x)
    probs = F.softmax(logits[0], dim=0)

    # Build dict of class -> probability (for Gradio Label)
    topk = torch.topk(probs, k=3)
    result = {classes[i]: float(topk.values[j])
              for j, i in enumerate(topk.indices)}
    return result

demo = gr.Interface(
    fn=predict,
    inputs=gr.Image(type="pil", label="Upload CIFAR-10 style image"),
    outputs=gr.Label(num_top_classes=3, label="Top-3 predictions"),
    title="CIFAR-10 MobileNetV2 Classifier",
)

if __name__ == "__main__":
    demo.launch(server_name="0.0.0.0", server_port=7861)
```

VI .RESULT

6.1 Training:

CPU Training: Serial

```

Device: cpu
Downloading CIFAR-10 (if needed)...
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
Train samples: 50000
Test samples: 10000
Total parameters: 2236682
Starting serial training...

Epoch 1/20 Time: 1551.65s Train Loss: 0.5516 Train Acc: 0.8131 Test Loss: 0.2555 Test Acc: 0.9136
Epoch 2/20 Time: 1546.19s Train Loss: 0.2941 Train Acc: 0.8992 Test Loss: 0.2087 Test Acc: 0.9285
Epoch 3/20 Time: 1566.97s Train Loss: 0.2404 Train Acc: 0.9171 Test Loss: 0.1859 Test Acc: 0.9372
Epoch 4/20 Time: 1543.63s Train Loss: 0.2009 Train Acc: 0.9320 Test Loss: 0.1605 Test Acc: 0.9463
Epoch 5/20 Time: 1575.07s Train Loss: 0.1761 Train Acc: 0.9383 Test Loss: 0.1692 Test Acc: 0.9430
Epoch 6/20 Time: 1538.95s Train Loss: 0.1590 Train Acc: 0.9441 Test Loss: 0.1634 Test Acc: 0.9453
Epoch 7/20 Time: 1541.39s Train Loss: 0.1424 Train Acc: 0.9507 Test Loss: 0.1682 Test Acc: 0.9464
Epoch 8/20 Time: 1553.15s Train Loss: 0.1266 Train Acc: 0.9550 Test Loss: 0.1612 Test Acc: 0.9477
Epoch 9/20 Time: 1560.95s Train Loss: 0.1184 Train Acc: 0.9589 Test Loss: 0.1564 Test Acc: 0.9519
Epoch 10/20 Time: 1544.80s Train Loss: 0.1124 Train Acc: 0.9607 Test Loss: 0.1530 Test Acc: 0.9527
Epoch 11/20 Time: 1543.50s Train Loss: 0.0760 Train Acc: 0.9740 Test Loss: 0.1329 Test Acc: 0.9572
Epoch 12/20 Time: 1541.92s Train Loss: 0.0629 Train Acc: 0.9792 Test Loss: 0.1309 Test Acc: 0.9586
Epoch 13/20 Time: 1543.24s Train Loss: 0.0572 Train Acc: 0.9810 Test Loss: 0.1308 Test Acc: 0.9583
Epoch 14/20 Time: 1538.86s Train Loss: 0.0516 Train Acc: 0.9831 Test Loss: 0.1305 Test Acc: 0.9591
Epoch 15/20 Time: 1539.75s Train Loss: 0.0488 Train Acc: 0.9841 Test Loss: 0.1268 Test Acc: 0.9617
Epoch 16/20 Time: 1540.70s Train Loss: 0.0456 Train Acc: 0.9851 Test Loss: 0.1279 Test Acc: 0.9613
Epoch 17/20 Time: 1549.03s Train Loss: 0.0464 Train Acc: 0.9844 Test Loss: 0.1316 Test Acc: 0.9610
Epoch 18/20 Time: 1543.67s Train Loss: 0.0449 Train Acc: 0.9848 Test Loss: 0.1334 Test Acc: 0.9600
Epoch 19/20 Time: 1541.59s Train Loss: 0.0417 Train Acc: 0.9864 Test Loss: 0.1319 Test Acc: 0.9602
Epoch 20/20 Time: 1547.84s Train Loss: 0.0414 Train Acc: 0.9856 Test Loss: 0.1308 Test Acc: 0.9612

Best test accuracy: 0.9617
Total training time: 30955.22s (515.92 min)
Saved best_mobilenetv2_cifar10_224.pth

```

GPU Training : 1GPU

```
Device: cuda
Downloading CIFAR-10 (if needed)...
Files already downloaded and verified
Files already downloaded and verified
Train samples: 50000
Test samples: 10000
Total parameters: 2236682
Starting serial training...

Epoch 1/20 Time: 570.94s Train Loss: 0.5879 Train Acc: 0.8007 Test Loss: 0.2834 Test Acc: 0.9027
Epoch 2/20 Time: 534.95s Train Loss: 0.2924 Train Acc: 0.8993 Test Loss: 0.2128 Test Acc: 0.9268
Epoch 3/20 Time: 535.08s Train Loss: 0.2302 Train Acc: 0.9201 Test Loss: 0.1831 Test Acc: 0.9374
Epoch 4/20 Time: 534.51s Train Loss: 0.1950 Train Acc: 0.9326 Test Loss: 0.1668 Test Acc: 0.9437
Epoch 5/20 Time: 550.41s Train Loss: 0.1708 Train Acc: 0.9415 Test Loss: 0.1651 Test Acc: 0.9440
Epoch 6/20 Time: 528.68s Train Loss: 0.1502 Train Acc: 0.9482 Test Loss: 0.1611 Test Acc: 0.9472
Epoch 7/20 Time: 549.55s Train Loss: 0.1326 Train Acc: 0.9542 Test Loss: 0.1580 Test Acc: 0.9484
Epoch 8/20 Time: 527.93s Train Loss: 0.1205 Train Acc: 0.9579 Test Loss: 0.1546 Test Acc: 0.9505
Epoch 9/20 Time: 527.14s Train Loss: 0.1129 Train Acc: 0.9609 Test Loss: 0.1477 Test Acc: 0.9511
Epoch 10/20 Time: 527.91s Train Loss: 0.1020 Train Acc: 0.9632 Test Loss: 0.1474 Test Acc: 0.9498
Epoch 11/20 Time: 527.42s Train Loss: 0.0718 Train Acc: 0.9752 Test Loss: 0.1311 Test Acc: 0.9570
Epoch 12/20 Time: 528.10s Train Loss: 0.0616 Train Acc: 0.9794 Test Loss: 0.1294 Test Acc: 0.9570
Epoch 13/20 Time: 527.89s Train Loss: 0.0588 Train Acc: 0.9804 Test Loss: 0.1280 Test Acc: 0.9590
Epoch 14/20 Time: 528.48s Train Loss: 0.0566 Train Acc: 0.9815 Test Loss: 0.1304 Test Acc: 0.9582
Epoch 15/20 Time: 556.74s Train Loss: 0.0554 Train Acc: 0.9810 Test Loss: 0.1288 Test Acc: 0.9577
Epoch 16/20 Time: 528.65s Train Loss: 0.0510 Train Acc: 0.9833 Test Loss: 0.1284 Test Acc: 0.9577
Epoch 17/20 Time: 528.39s Train Loss: 0.0512 Train Acc: 0.9831 Test Loss: 0.1280 Test Acc: 0.9578
Epoch 18/20 Time: 527.72s Train Loss: 0.0492 Train Acc: 0.9838 Test Loss: 0.1298 Test Acc: 0.9583
Epoch 19/20 Time: 527.82s Train Loss: 0.0484 Train Acc: 0.9843 Test Loss: 0.1277 Test Acc: 0.9588
Epoch 20/20 Time: 527.99s Train Loss: 0.0453 Train Acc: 0.9850 Test Loss: 0.1277 Test Acc: 0.9603

Best test accuracy: 0.9603
Total training time: 10698.08s (178.30 min)
Saved best_mobilenetv2_cifar10_224.pth
```

Distributed Training (MPI+DDP) : 2 GPU

```
=====
MPI + DDP MobileNetV2 CIFAR-10 (224x224)
=====
Backend: nccl, World size: 2, Device(rank0): cuda:0
Downloading CIFAR-10 (if not present)...
Files already downloaded and verified
Files already downloaded and verified
Train samples: 50000
Test samples: 10000
Total parameters: 2236682
Starting distributed training...

Epoch 1/20 Time: 287.58s Train Loss: 0.7229 Test Loss: 0.3092 Test Acc(local): 0.8934
Epoch 2/20 Time: 253.23s Train Loss: 0.3278 Test Loss: 0.2322 Test Acc(local): 0.9178
Epoch 3/20 Time: 253.73s Train Loss: 0.2613 Test Loss: 0.2124 Test Acc(local): 0.9250
Epoch 4/20 Time: 282.35s Train Loss: 0.2191 Test Loss: 0.1865 Test Acc(local): 0.9358
Epoch 5/20 Time: 253.07s Train Loss: 0.1923 Test Loss: 0.1731 Test Acc(local): 0.9380
Epoch 6/20 Time: 253.20s Train Loss: 0.1666 Test Loss: 0.1628 Test Acc(local): 0.9438
Epoch 7/20 Time: 282.16s Train Loss: 0.1489 Test Loss: 0.1719 Test Acc(local): 0.9376
Epoch 8/20 Time: 281.62s Train Loss: 0.1360 Test Loss: 0.1567 Test Acc(local): 0.9450
Epoch 9/20 Time: 259.59s Train Loss: 0.1227 Test Loss: 0.1550 Test Acc(local): 0.9510
Epoch 10/20 Time: 253.76s Train Loss: 0.1126 Test Loss: 0.1537 Test Acc(local): 0.9490
Epoch 11/20 Time: 253.00s Train Loss: 0.0887 Test Loss: 0.1395 Test Acc(local): 0.9526
Epoch 12/20 Time: 257.89s Train Loss: 0.0803 Test Loss: 0.1389 Test Acc(local): 0.9538
Epoch 13/20 Time: 253.21s Train Loss: 0.0790 Test Loss: 0.1383 Test Acc(local): 0.9558
Epoch 14/20 Time: 253.12s Train Loss: 0.0755 Test Loss: 0.1355 Test Acc(local): 0.9548
Epoch 15/20 Time: 273.76s Train Loss: 0.0722 Test Loss: 0.1352 Test Acc(local): 0.9552
Epoch 16/20 Time: 255.02s Train Loss: 0.0721 Test Loss: 0.1380 Test Acc(local): 0.9548
Epoch 17/20 Time: 253.76s Train Loss: 0.0695 Test Loss: 0.1355 Test Acc(local): 0.9558
Epoch 18/20 Time: 253.76s Train Loss: 0.0691 Test Loss: 0.1344 Test Acc(local): 0.9552
Epoch 19/20 Time: 253.26s Train Loss: 0.0676 Test Loss: 0.1381 Test Acc(local): 0.9540
Epoch 20/20 Time: 253.27s Train Loss: 0.0652 Test Loss: 0.1389 Test Acc(local): 0.9532

Best local test accuracy: 0.9558
Total training time: 5220.57s (87.01 min)
Saved best_mobilenetv2_cifar10_224_mpi.pth
```

6.2 Performance & Accuracy Analysis Table (CIFAR-10)

Execution Mode	Hardware Used	Training Type	Total Time (Minutes)	Best test accuracy
CPU	Single CPU	Single Process	515.92	0.9617
GPU	Single GPU	Single Process	178.30	0.9603
Distributed GPU (MPI)	Multiple(2) GPUs	Data Parallel	87.01	0.9558

The experimental results show that while serial CPU execution is the slowest, GPU acceleration significantly improves training time, and distributed GPU training using MPI provides the highest performance improvement. Importantly, all execution modes achieve comparable accuracy , proving that distributed training enhances efficiency without compromising model performance.

Result of Gradio :



VII . CONCLUSION

This project demonstrates the practical impact of High-Performance Computing (HPC) on deep learning training by leveraging GPU acceleration and MPI-based distributed computing. Training on a serial CPU required 515.92 minutes, while the use of a single GPU reduced execution time to 178.30 minutes, achieving a $2.9\times$ speedup through parallel tensor processing. Further performance gains were achieved using MPI on multiple GPUs, reducing training time to 87.01 minutes and resulting in an overall speedup of nearly $6\times$. These results clearly highlight the scalability and efficiency of HPC systems in handling computationally intensive deep learning workloads. Additionally, the trained model was deployed using Gradio for real-time inference, demonstrating how HPC-trained models can be effectively integrated into practical, user-facing applications.

VIII. Future Enhancement

Species-Level Classification: Extend the current CIFAR-10–based model to classify specific animal species and plant species.

Dataset Expansion: Incorporate large-scale biodiversity datasets such as wildlife image repositories to improve model generalization and real-world applicability.

AI-based Issue Prioritization: Integrate deeper and more powerful architectures (ResNet, EfficientNet, Vision Transformers) to improve accuracy for complex species-level classification tasks.

IX . REFERENCES

1. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 4510-4520. <https://doi.org/10.1109/CVPR.2018.00474>
2. Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., ... & He, K. (2017). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677*. <https://arxiv.org/abs/1706.02677>
3. Lin, Y., Han, S., Mao, H., Wang, Y., & Dally, W. J. (2017). Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. *arXiv preprint arXiv:1712.01887*. <https://arxiv.org/abs/1712.01887>
4. Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2018). AutoAugment: Learning Augmentation Policies from Data. *arXiv preprint arXiv:1805.09501*. <https://arxiv.org/abs/1805.09501>
5. Zhong, Z., Zheng, L., Kang, G., Li, S., & Yang, Y. (2020). Random Erasing Data Augmentation. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 13001-13010. <https://doi.org/10.1109/CVPR.2019.01249>
6. Forum, M. P. I. (2015). MPI: A Message-Passing Interface Standard Version 3.1. *High Performance Computing Center Stuttgart (HLRS)*.
7. Yosinski, J., Clune, J., Bengio, Y., & Liphardt, H. (2014). How transferable are features in deep neural networks?. *Advances in Neural Information Processing Systems*, 27, 3320-3328. <https://arxiv.org/abs/1411.1792>
8. Abid, A., Abdulkader, A., & Abid, A. (2021). Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild. *arXiv preprint arXiv:2109.02295*. <https://arxiv.org/abs/2109.02295>
9. PyTorch Distributed Documentation: <https://pytorch.org/docs/stable/distributed.html>
10. CIFAR-10 Dataset: Krizhevsky, A., & Hinton, G. (2009). Learning Multiple Layers of Features from Tiny Images. *Technical Report*.
11. torchvision Models: <https://pytorch.org/vision/stable/models.html>
12. Gradio Documentation: <https://www.gradio.app/>