Deep Learning for Edge AI: MobileNetV2 CNN Training over ARM-based Clusters

Dimitrios Papakyriakou
PhD Candidate
Department of Electronic Engineering
Hellenic Mediterranean University
Crete, Greece

Ioannis S. Barbounakis
Assistant Professor
Department of Electronic Engineering
Hellenic Mediterranean University
Crete, Greece

ABSTRACT
This paper presents a comprehensive investigation into the strong scaling performance of distributed training for Convolutional Neural Networks (CNNs) using the MobileNetV2 architecture on a resource-constrained Beowulf cluster composed of 24 Raspberry Pi 4B nodes (8 GB RAM each). The training system employs the Message Passing Interface (MPI) via MPICH with synchronous data parallelism, running two processes per node across 2 to 48 total MPI processes. A fixed CIFAR-10 dataset was used, and all experiments were standardized to 10 epochs to maintain memory stability.

The study jointly evaluates execution time scaling, training/test accuracy, and convergence loss to assess both computational performance and learning quality under increasing parallelism. Training time decreased nearly ten-fold at cluster scale, reaching a maximum speedup of (≈9.99×) with (≈41.6 %) parallel efficiency at 48 processes. Efficiency remained very high at small scales (≈90.9 % at np=4) and moderate at np=8 (≈52.3 %), confirming that MPI scaling itself is effective up to this range. However, while single-node and small-scale runs (up to 4–8 MPI processes) preserved strong generalization ability, larger scales suffered from sharply reduced per-rank dataset sizes, causing gradient noise and eventual collapse of test accuracy to the random guess baseline (10 %).

These results demonstrate that, although ARM-based Raspberry Pi clusters can support feasible small-scale distributed deep learning, strong scaling beyond an optimal process count leads to “fast-but-wrong” training in which wall-clock performance improves but model utility on unseen data is lost. This work provides the first detailed end-to-end evaluation of MPI-based synchronous CNN training across an ARM-based edge cluster, and outlines future research including comparative scaling with SqueezeNet and exploration of ultra-low-power Spiking Neural Networks (SNNs) for neuromorphic edge learning.

Keywords
Convolutional Neural Networks (CNNs), Distributed Deep Learning, Beowulf Cluster, ARM Architecture, Raspberry Pi Cluster, Parallel Computing, Message Passing Interface (MPI), MPICH, Low-Cost Clusters, Distributed Systems, HPC, MobileNet, Edge Computing, Parallel Efficiency, Edge Deep Learning.

1. INTRODUCTION
Deep learning has emerged as a cornerstone of modern artificial intelligence, with Convolutional Neural Networks (CNNs) demonstrating state-of-the-art performance in image recognition, classification, and embedded vision systems [1]. The training of CNN models, however, remains computationally intensive and is typically performed on centralized, high-performance GPU clusters. In contrast, recent developments in edge computing and distributed systems have prompted growing interest in enabling scalable training on low-cost, heterogeneous platforms.

This study focuses on the distributed training of MobileNet, a computationally efficient CNN architecture designed for mobile and embedded environments [2]. The implementation employs the Message Passing Interface (MPI) on a Beowulf cluster consisting of 24 Raspberry Pi 4B nodes, each equipped with an ARM-based processor and 8GB of RAM. A total of 48 MPI processes are executed - two per node - using mpi4py to facilitate data-parallel training over a fixed CIFAR-10 dataset and to manage inter-process communication in a synchronous training model [3]. The objective is to evaluate strong scaling behavior in this resource-constrained environment by examining how model convergence, classification accuracy, and total training time are affected as the number of parallel processes increases. Due to hardware memory limitations, the number of training epochs is fixed at (10), beyond which execution becomes unstable. This constraint reflects the practical limitations faced by edge devices and embedded systems, where computational capacity and memory bandwidth are restricted [4]. The experimental results aim to provide insight into the scalability of deep learning workloads on ARM-based multi-node systems. Particular attention is given to the impact of parallelism on training performance and communication overhead, offering empirical evidence relevant to the deployment of distributed CNN training in constrained environments such as fog and edge computing platforms [5]

The Raspberry Pi 4 Model B (8GB RAM), illustrated in "Figure 1", serves as the fundamental building block of the cluster. Each unit features a 64-bit quad-core ARMv8 Cortex-A72 CPU clocked at 1.5 GHz [6], [7]. Its affordability and accessibility were key factors in its selection as the base hardware for constructing a high-performance computing cluster, enabling a systematic evaluation of its capabilities in parallel processing and distributed computing environments.

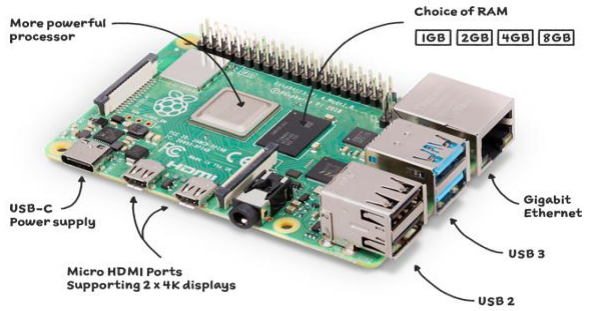

Figure 1: Single Board Computer (SBC) - Raspberry Pi 4 Model B [6], [7].

2. SYSTEM DESCRIPTION

2.1 Hardware Equipment

The computational platform used for this study consists of a cost-effective yet capable Beowulf-style cluster constructed from 24 Raspberry Pi 4B units, each equipped with 8GB of RAM. One device is configured as the master node, responsible for resource allocation and process orchestration, while the remaining 23 function as worker nodes participating in parallel execution under MPI coordination. An overview of the physical deployment is presented in "Figure 2". The cluster is physically organized into four vertical stacks, with each stack comprising six Raspberry Pi boards. All nodes are interconnected through TP-Link TL-SG1024D unmanaged Gigabit Ethernet switches, supporting up to 1 Gbps of bandwidth per device. This network architecture ensures consistent and low-latency inter-node communication, effectively simulating a high-performance computing environment within an ARM-based embedded framework.

To guarantee consistent power delivery across all units, the system employs two industrial-grade switch-mode power supplies (60A, 5V), adjusted to output 5.80V. This voltage tuning mitigates power loss across extended cabling paths and ensures reliable operation under sustained parallel workloads. For data storage, the master node utilizes a 1TB Samsung 980 PCIe 3.0 NVMe SSD, while each worker node is fitted with a 256GB Patriot P300 NVMe M.2 SSD. These storage components provide high-throughput local I/O performance and enable smooth data handling during training, especially when working with larger input datasets and intermediate model states.

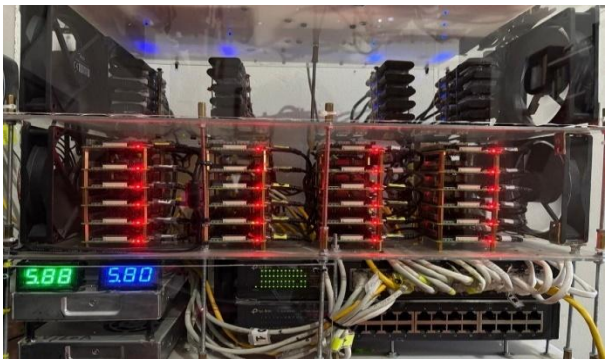

Figure 2: Deployment of the Beowulf Cluster with (24) RPi-4B (8GB).

2.2 Software Environment and Toolchain

The software stack employed in this study was designed to support distributed training of deep neural networks over ARM-based systems. All nodes in the cluster operated under Raspberry Pi OS 64-bit Lite (Debian-based), running Python 3.11.5 in a shared virtual environment mounted via NFS to ensure consistency across the system. Communication between processes was managed through MPICH v4.2.0 and coordinated using the mpi4py interface (v3.1.6).

In addition to the numerical and scientific computing libraries used in previous benchmarking efforts - namely NumPy (v1.26.4), SciPy (v1.13.0), scikit-learn (v1.4.2), and psutil (v7.0.0) - this project introduced deep learning-specific dependencies, including TensorFlow (v2.15.0), PyTorch (2.8.0) and Keras (integrated within TensorFlow) to support CNN construction, training, and evaluation.

All packages were compiled for compatibility with the ARMv8 architecture and deployed uniformly across the cluster to ensure reproducibility and eliminate version-related inconsistencies. The Python virtual environment was shared across all nodes using NFS, while passwordless SSH and synchronized environmental variables (including LD_LIBRARY_PATH and UCX_TLS) facilitated seamless MPI-based coordination for training execution.

All MPI executions were performed within the virtual environment using mpiexec, with explicit *machinefile* configuration and core binding to ensure optimal resource utilization per process. This containerized configuration not only guarantees identical behavior across all nodes, but also enhances the scientific reliability and reproducibility of experimental results.

2.3 Design

The physical and logical structure of the Raspberry Pi cluster used in this study is presented in "Figure 2" and "Figure 3" respectively. The system comprises 24 Raspberry Pi 4B nodes, each equipped with 8GB of RAM. All nodes are interconnected via a 24-port Gigabit Ethernet switch, enabling full-duplex communication with bandwidth capacities of up to 1 Gbps per device. Within this architecture, one node operates as the master (head) node, managing scheduling and coordination tasks, while the remaining 23 nodes function as distributed compute resources. Each unit is assigned a static IP address to facilitate deterministic routing, and all inter-node communication is secured using SSH.

The master node is provisioned with a 1TB Samsung 980 PCIe 3.0 NVMe M.2 SSD, capable of achieving up to 3500 MB/s read and 3000 MB/s write speeds under ideal conditions. Each worker node is equipped with a 256GB Patriot P300 NVMe M.2 SSD, offering up to 1700 MB/s read and 1100 MB/s write speeds. These drives are interfaced through USB 3.0 connections, which support theoretical transfer rates up to 4.8 Gbps (600 MB/s). This configuration represents a substantial performance improvement over traditional USB 2.0 or microSD-based systems, offering significantly faster I/O throughput for model data and dataset access.

While the USB 3.0 interface does not reach the bandwidth of native PCIe lanes, the integration of NVMe-class SSDs dramatically enhances storage responsiveness and workload execution times. This contributes to a measurable performance uplift in training and testing scenarios involving large-volume parameter updates or frequent disk access. The ability of the Raspberry Pi 4B platform to boot and operate reliably from external SSDs further supports this design choice.

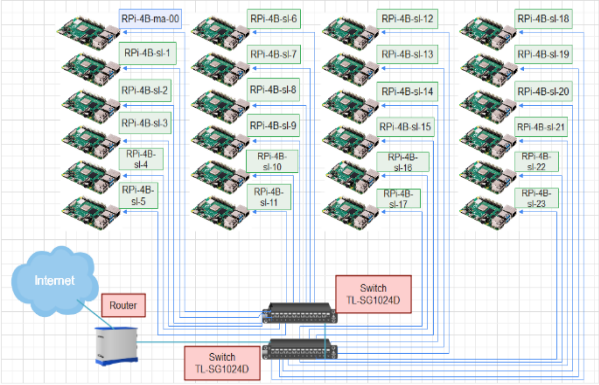

Figure 3: RPi-4B Beowulf cluster architecture diagram.

- *Target Platform and Feasibility Considerations*

  This study focuses on evaluating the feasibility and performance of distributed convolutional neural network (CNN) training using the MobileNet architecture on a resource-constrained Beowulf cluster composed of Raspberry Pi 4B devices with 8GB of RAM per node. While MobileNet was originally designed for lightweight image classification tasks on mobile and embedded systems, training such models on low-power clusters presents unique challenges and opportunities for benchmarking.

  The target cluster features ARM Cortex-A72 processors, lacking hardware accelerators such as GPUs or vectorized instruction sets like AVX. Consequently, all computations are executed on CPUs, which naturally results in limited training speed compared to high-performance computing environments. Furthermore, inter-node communication is performed via Ethernet connections, introducing additional latency and bandwidth constraints during distributed training operations.

  Despite these hardware limitations, MobileNet remains an ideal candidate for experimentation due to its lightweight design based on depth wise separable convolutions and its moderate memory footprint. By leveraging small-scale datasets such as *CIFAR-10*, we ensure that training remains computationally feasible within the available memory (8GB per node) and processing capabilities of the cluster.

  The main objective of this study is not to achieve state-of-the-art training speed or accuracy but rather to explore the viability, performance scaling, and communication overhead of distributed deep learning workloads on low-power, cost-effective cluster architectures. This approach offers valuable insights for edge computing scenarios, where similar hardware and performance constraints are common.

2.4 Theoretical Background: The MobileNet CNNs Clustering Algorithm

Convolutional Neural Networks (CNNs) represent a fundamental class of deep learning architectures that have demonstrated exceptional performance in image classification tasks due to their ability to automatically extract hierarchical spatial features from data [1]. Among the many CNN variants, *MobileNet* has emerged as a prominent architecture optimized for mobile and embedded environments. It achieves a favorable balance between performance and efficiency through the use of depth wise separable convolutions -a factorization technique that decomposes standard convolutions into two simpler operations, substantially reducing both parameter count and computational cost- [2].

The version employed in this study is *MobileNetV2*, which has been shown to maintain competitive classification accuracy while dramatically lowering the number of floating-point operations per inference. Its compact size and low memory footprint make it particularly suitable for ARM-based embedded systems, such as the Raspberry Pi 4B platform used in this research [2], [8].

To evaluate the training behavior of *MobileNet* in a distributed setting, the model is trained on the *CIFAR-10* dataset -a widely adopted image classification benchmark consisting of 60,000 color images (32×32 resolution) evenly divided across 10 classes [9]. The dataset contains 50,000 training images and 10,000 test images and is well-suited for performance evaluations on constrained hardware due to its moderate size and manageable complexity.

Although more challenging datasets such as *CIFAR-100* exist, the hardware limitations of the ARM-based Raspberry Pi cluster pose significant barriers to their use. *CIFAR-100* increases the number of output classes tenfold, which in turn inflates the final dense layers of the model, resulting in higher memory consumption and greater computational demand during training. Empirical tests demonstrated that training *MobileNet* on *CIFAR-100* caused memory exhaustion or performance degradation beyond sustainable levels -even when batch sizes were minimized. By contrast, *MobileNet* was successfully trained on *CIFAR-10* up to *10 epochs* without exceeding memory or thermal constraints, confirming its feasibility on this cluster configuration.

Given these constraints, *CIFAR-10* was selected to ensure training stability while still allowing analysis of learning performance and distributed scaling behavior. The dataset's compact structure aligns well with the goal of benchmarking MobileNet's efficiency in parallel execution on low-power embedded systems.

3. METHODOLOGY

3.1 System Configuration and Experimental Context

The experiments were conducted on a custom-built Beowulf cluster comprising 24 Raspberry Pi 4B nodes. Each node is equipped with a quad-core ARM Cortex-A72 CPU and 8GB of LPDDR4 RAM, and interconnected via Gigabit Ethernet to facilitate low-latency inter-node communication. The cluster's low-power architecture, limited memory footprint, and absence of hardware accelerators such as GPUs or AVX-capable CPUs present a unique set of constraints that directly influence the design of the training pipeline.

To accommodate these limitations, the study leverages the *MobileNetV2* convolutional neural network - a lightweight model designed for mobile and embedded devices. Its use of depthwise separable convolutions reduces the number of trainable parameters and computational load, making it particularly well-suited to the ARM-based environment.

The *CIFAR-10* dataset is used for all training experiments. It comprises 60,000 color images (32×32 resolution) across 10 classes, with 50,000 used for training and 10,000 for testing. Its compact size aligns well with the available RAM and I/O

capacity of the cluster, enabling full dataset loading in memory without inducing swapping or resource contention.

All nodes run Raspberry Pi OS 64-bit Lite with Python 3.11.5 in a shared virtual environment mounted via NFS. The single-node batch-size sweep uses a lightweight *PyTorch 2.8.0* driver (with torchvision) configured to the same MobileNetV2 family, input preprocessing, optimizer, and synchronous MPI pattern (gather/average/broadcast) to accelerate parameter exploration.

The distributed full-cluster training uses *TensorFlow 2.15 (Keras API)* with the same synchronous data-parallel scheme via mpi4py 3.1.6 over MPICH 4.2.0. Unless otherwise stated, both stacks use MobileNetV2 (width multiplier 0.5), channels-last (NHWC) tensors resized to (128×128), SGD (lr=5e-4, momentum=0.9, weight decay=0), and identical normalization. All packages are CPU-only builds compiled for ARMv8 and distributed uniformly across nodes. MPI processes are launched with mpiexec using an explicit machinefile and core binding for hardware-locality reproducibility. Because both stacks fix the same architecture (MobileNetV2, width 0.5), input shape/layout (128×128, NHWC), normalization, optimizer and hyperparameters (SGD, lr = 5e-4, momentum = 0.9, no weight decay), and the same synchronous MPI update pattern (gather → average → broadcast), the PyTorch sweep and TensorFlow/Keras runs are directly comparable in compute/memory footprint and training dynamics.

NHWC is a tensor layout that orders image data as (N: batch size), (H: image height), (W: image width), (C: number of channels), so a 32-image batch of 128×128 RGB pictures is shaped (32, 128, 128, 3) in NHWC. On ARM CPUs, NHWC can improve cache/NEON access patterns—especially for depthwise separable convs (like MobileNet)—so we enable it to squeeze a bit more throughput.

3.2 Distributed Training Strategy

It's adopted data parallelism: each MPI process (rank) trains on a *disjoint shard* of the dataset and contributes to synchronous model updates at defined barriers. It's run two ranks per Raspberry Pi 4B to balance CPU cores and 8 GB RAM per node.

3.2.1 Deterministic sharding

Let (N_p) be the total number of ranks (processes). During each epoch we partition the 50,000 CIFAR-10 training samples by strided indexing (i.e *sample (i) is assigned to* $rank\ (k)\ if (i)\ mod\ N_p = k.$ Thus, each rank processes approximately ($\approx$ *50,000/*N_p) unique samples per epoch. This guarantees non-overlapping, reproducible data splits and ensures a uniform workload across all ranks.

$$shard_r = \{\, i \in \{1 \ldots.,50000\} \mid i \ mod\ N_p = r\},$$

so that each rank $r \in \{0, \ldots, N_p - 1\}$ processes $\approx \frac{50000}{N_p}$ samples per epoch $e.g\ 25000\ for\ N_p = 2;\ \approx 1042\ for\ N_p = 48$. This guarantees non-overlapping, reproducible data splits and uniform load.

3.2.2 Synchronous SGD via MPI collectives

Within an epoch each rank performs standard forward/backward passes on its shard. At the epoch boundary we emulate synchronous SGD: all ranks send their model parameters (or updates) to the root using *comm.gather(),* the root averages them, and the averaged weights are broadcast back with *comm.bcast().* (Functionally this is equivalent to an *Allreduce* on parameters; we keep the gather/broadcast pattern for transparency and easier logging.) The root also runs the final test evaluation.

3.2.3 Per-node batch-size profiling (empirical sweep)

Because batch size primarily governs local memory footprint and CPU utilization per rank, we selected it via a per-node sweep on one RPi (2 ranks/node) and spot-checked on two additional RPi's to rule out hardware heterogeneity.

- *Experimental Protocol:* The experiment was performed for each batch size b ∈ {8, 16, 24, 32} images per node (i.e., b/2 per rank). Logged metrics included: median epoch time, peak per-process RSS (via *psutil*), node-level RAM and swap usage (*free -m, vmstat*), CPU utilization (*mpstat*), and OOM events (*dmesg*).

- *Outcome*: The sweep identified batch-per-node = 24 (equivalently 12 samples per rank for 2 ranks/node) as the best time–stability trade-off. This setting maintained high CPU utilization without invoking swap and produced the lowest median epoch time. Larger batches (≥ 24) intermittently induced page-cache pressure and slower epoch times, while smaller batches (e.g., 8) under-utilized the CPU. Replication on two additional RPi nodes confirmed the same regime, with variation within only a few percent.

3.2.4 Fixed benchmarking hyperparameters

To ensure consistent and controlled benchmarking across all cluster sizes (N_p) the following hyperparameter values were used:

- *Epochs*: 10 (upper bound before memory/thermal instability on sustained runs).

- *Batch size*: 24 images per node (thus 12 per rank with 2 ranks/node).

- *Learning rate*: 0.0005.

  Why LR = 0.0005: The training was performed with synchronous SGD (Synchronous Stochastic Gradient) under small effective batch (12 images/rank). Small batches increase gradient noise; a conservative LR improves stability and avoids oscillations. Empirically, a $\{0.0003,\ 0.0005,\ 0.001\}$, showed 0.0005 delivered the fastest *stable* validation-loss decrease without divergence on CPU-only nodes.
  This choice is also consistent with common practice for MobileNet/CIFAR-10 when training from scratch with small batches and SGD [10], [11]. Typical starting LRs fall in $(10^{-4} - 10^{-3})$ and are adjusted downward as batch size decreases.

3.2.5 Fixed benchmarking hyperparameters in one RPi (pi@rpi4B-ma-00)

Before the main MobileNet CNN benchmarking was performed on the complete cluster, a dedicated series of experiments was carried out to fine-tune the batch size (i.e., the number of images $number\ of\ images \in \{8, 16, 24, 32\}$ processed per node in each step). By systematically sweeping batch sizes and analyzing metrics such as median epoch time, CPU and memory utilization, and

process stability, we established the optimal configuration. This enabled all subsequent distributed training runs to use the empirically "best" batch size identified for time–stability trade-off and hardware efficiency.

The observations are the following based on "Figure 4", "Figure 5", "Table 1" considering eventually that the (batch size = 24) is the right choice for the RPi 4B (8GB) RAM:

- *Empirical optimum in throughput*: The sweep shows a clear maximum 9.67 img/sec at 24 images/node. Throughput rises from 8→16→24 and then drops at 32.

- *Memory still comfortable:* Peak RSS increases as batch grows (≈ 638 → 640 → 718 → 806 MB per rank). With 2 ranks/node, that's (≈ 1.4–1.6 GB) per node—well below 8 GB, leaving room for OS, buffers, and MPI.

  RSS: (Resident Set Size). The amount of physical RAM a process is using at a moment—code, data, stack, plus any shared pages that are actually resident in memory (not swap).

  - Peak RSS = the highest RSS observed during the run.

  Note: *summing RSS across processes can overestimate total node usage because shared pages get counted in each process. For precise per-node memory, use cgroup metrics or tools like ps_mem/smem*

- *Why 32 degrades on RPi (ARM Cortex-A72):* At 32/node (16/rank) the working set of activations/gradients starts to exceed cache sweet spots. The A72's L2 and DRAM bandwidth become the bottleneck: more cache misses and memory traffic → step time balloons (3.555 s vs 2.482 s), lowering throughput despite the larger batch.

The justification why the batch size (24 images per node) is the best choice is summarized below:

- It maximizes compute density per step enough to amortize *Python*/*DataLoader* overhead without tipping into memory-bandwidth thrash.

- It keeps per-rank CPU saturated (≈ 100% Process CPU) and memory headroom healthy (≈ 718 MB/rank in this setup).

- It scales cleanly to the full cluster under data parallelism (2 ranks/node): global batch grows linearly with nodes while the per-rank working set stays stable—good for convergence and for keeping per-step overhead low at scale.

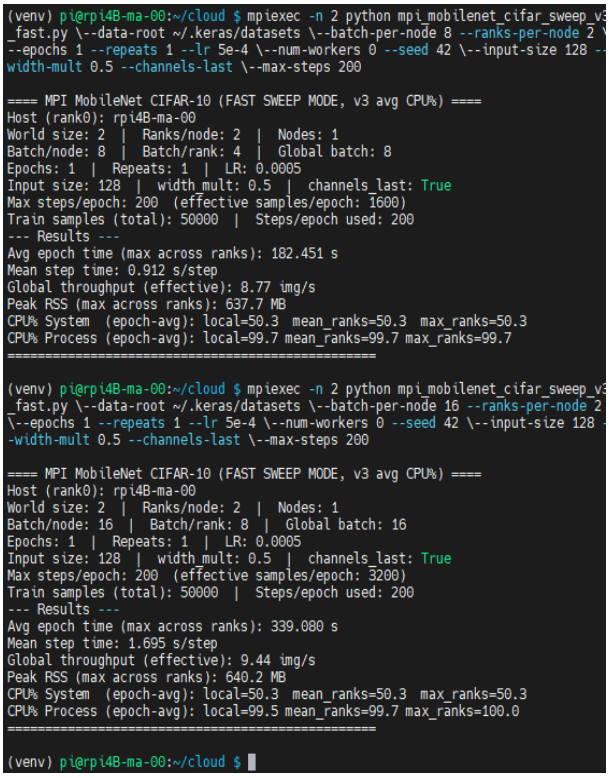

Figure 4: Visualization of Batch Size (8 and 16 images per node) Benchmarking Results "Table 1" for MPI MobileNetV2 on a Single RPi Node.

Table 1. MPI MobileNetV2 CIFAR-10, batch (Images) size benchmarking -best option survey-.

Batch /node	**Batch /rank**	**Avg. epoch time** (sec)	**Mean step** (sec)	**Global through -put** (img/s)	**Peak RSS** (MB)	**Avg. Proc -ess CPU (%)**
8	4	182.45	0.912	8.77	637.7	99.7
16	8	339.08	1.695	9.44	640.2	99.7
24	12	496.43	2.482	9.67	717.7	99.7
32	16	711.07	3.555	9.00	806.4	99.7

```
(venv) pi@rpi4B-ma-00:~/cloud $ mpiexec -n 2 python mpi_mobilenet_cifar_sweep_v3
_fast.py \--data-root ~/.keras/datasets \--batch-per-node 24 --ranks-per-node 2
\--epochs 1 --repeats 1 --lr 5e-4 \--num-workers 0 --seed 42 \--input-size 128 -
-width-mult 0.5 --channels-last \--max-steps 200

==== MPI MobileNet CIFAR-10 (FAST SWEEP MODE, v3 avg CPU%) ====
Host (rank0): rpi4B-ma-00
World size: 2  |  Ranks/node: 2  |  Nodes: 1
Batch/node: 24  |  Batch/rank: 12  |  Global batch: 24
Epochs: 1  |  Repeats: 1  |  LR: 0.0005
Input size: 128  |  width_mult: 0.5  |  channels_last: True
Max steps/epoch: 200  (effective samples/epoch: 4800)
Train samples (total): 50000  |  Steps/epoch used: 200
--- Results ---
Avg epoch time (max across ranks): 496.429 s
Mean step time: 2.482 s/step
Global throughput (effective): 9.67 img/s
Peak RSS (max across ranks): 717.7 MB
CPU% System  (epoch-avg): local=50.3  mean_ranks=50.3  max_ranks=50.3
CPU% Process (epoch-avg): local=99.9 mean_ranks=99.7 max_ranks=99.9
================================================

(venv) pi@rpi4B-ma-00:~/cloud $ mpiexec -n 2 python mpi_mobilenet_cifar_sweep_v3
_fast.py \--data-root ~/.keras/datasets \--batch-per-node 32 --ranks-per-node 2
\--epochs 1 --repeats 1 --lr 5e-4 \--num-workers 0 --seed 42 \--input-size 128 -
-width-mult 0.5 --channels-last \--max-steps 200

==== MPI MobileNet CIFAR-10 (FAST SWEEP MODE, v3 avg CPU%) ====
Host (rank0): rpi4B-ma-00
World size: 2  |  Ranks/node: 2  |  Nodes: 1
Batch/node: 32  |  Batch/rank: 16  |  Global batch: 32
Epochs: 1  |  Repeats: 1  |  LR: 0.0005
Input size: 128  |  width_mult: 0.5  |  channels_last: True
Max steps/epoch: 200  (effective samples/epoch: 6400)
Train samples (total): 50000  |  Steps/epoch used: 200
--- Results ---
Avg epoch time (max across ranks): 711.071 s
Mean step time: 3.555 s/step
Global throughput (effective): 9.00 img/s
Peak RSS (max across ranks): 806.4 MB
CPU% System  (epoch-avg): local=50.3  mean_ranks=50.3  max_ranks=50.3
CPU% Process (epoch-avg): local=99.8 mean_ranks=99.7 max_ranks=99.8
================================================

(venv) pi@rpi4B-ma-00:~/cloud $
```

Figure 5: Visualization of Batch Size (24 and 32 images per node) Benchmarking Results "Table 1" for MPI MobileNetV2 on a Single RPi Node

```
mpiexec -n 2 python mpi_mobilenet_cifar_sweep_v3_fast.py \
  --data-root ~/.keras/datasets \
  --batch-per-node {8|16|24|32} \
  --ranks-per-node 2 \
  --epochs 1 --repeats 1 \
  --lr 5e-4 --num-workers 0 --seed 42 \
  --input-size 128 --width-mult 0.5 --channels-last \
  --max-steps 200
```

Figure 6: Benchmarking hyperparameters: Command Template, fixed flags.

"Figure 6", explains briefly the fixed flags in the command used to run the testing:

--batch-per-node: total images processed per step by the node (split evenly across ranks).

--ranks-per-node: MPI processes per RPi (here 2).

--max-steps 200: cap steps/epoch to speed up the sweep (we measure steady-state without running the full 50 k samples).

--width-mult 0.5: half-width MobileNetV2 (lighter model suited to ARM).

--input-size 128, *--channels-last*: resize CIFAR-10 to 128×128; NHWC layout—both choices help MobileNet and ARM memory access patterns.

--num-workers 0: single-threaded data loading—safer on RPi to avoid context-switch overhead and memory spikes.

--lr 5e-4, *--seed 42*, *--repeats 1*, *--epochs 1*: standard control knobs for learning rate, reproducibility, and sweep brevity.

Results Meaning:

- *Avg epoch time* (max across ranks): wall-clock time set by the slowest rank—appropriate for synchronous training.
- *Mean step time*: Avg epoch time / (steps used). Lower is better.
- *Global throughput*: effective images/second processed by the node. Higher is better for selecting batch size.
- *Peak RSS*: max resident memory (per rank). We want ample headroom relative to 8 GB/node.
- *CPU%*: epoch-averaged utilization; Process ≈100% confirms the cores running the ranks are saturated.

Note: *"Table 1", shows Process CPU ≈ 100%, whereas "Figure 4", "Figure 5" show additional info such as Process CPU and System CPU ≈ 50%. System CPU (≈ 50%) means that half of the CPU time is being consumed by kernel-level work since 2 cores (out of 4) are involved in the training process (2 MPI processes).*

As a conclusion, for *MobileNetV2* (width 0.5, 128×128) on RPi 4B (8 GB) with 2 ranks/node and threads=1, batch-per-node = 24 offers the best trade-off: *highest throughput*, *safe memory*, and *fully saturated training cores*. Batch = 16 is acceptable but slower; batch = 32 shows clear diminishing returns and higher memory pressure.

3.3 Strong Scaling Experimental Design

The core experimental objective is to analyse the strong scaling performance of MobileNet training under distributed data-parallel execution. In a strong scaling scenario, the total workload (i.e., the dataset and model) remains fixed while the number of computational processes increases. The study examines how overall training time, convergence behaviour, and parallel efficiency evolve as resources scale.

MPI configurations tested are:

$$np \in \{2, 4, 8, 16, 24, 32, 40, 48\}$$

where (*np*) denotes the number of MPI processes, and each value corresponds to $p = \frac{np}{2}$ Raspberry Pi nodes, given the *2-process-per-node* configuration.

Key performance metrics include:

- *Mean Train Accuracy Across Ranks:*

$$Train\ Accuracy_{\ mean} = \frac{1}{N}\sum_{i=0}^{N-1} Accuracy_{rank\ i}$$

Where:
$N = number\ of\ ranks \in \{2, 4, 8, 16, 24, 32, 40, 48\}$

$Accuracy_{rank\ i}$
$= reported\ accuracy\ per\ rank\ in\ the\ end\ of\ epoch\ 10$

- *Total training time for 10 epochs across all (np) values.*
- *Speedup S(np):*

$$S(np) = \frac{T_{base}}{T_{np}}$$

Where (*Tbase)* is the runtime for the baseline case (np = 2)

- *Parallel Efficiency E(np):*

$$E(np) = \frac{S_{(np)}}{\frac{np}{2}} x\ 100\%$$

Note:
Efficiency is reported relative to a baseline of np=2 MPI processes.

For instance, the configuration with 4 MPI processes achieved an efficiency of 90.92% of the ideal linear scaling, indicating relatively high parallel performance before significant communication overhead emerged "Table 2", "Figure 10".

- *Convergence metrics:* training and validation loss, accuracy per epoch.

- *Communication overhead*, inferred from increased time variance or efficiency drop at high process counts (e.g., np ≥ 32).

All experiments were conducted in isolation to avoid network interference or thermal throttling, and each configuration was repeated for consistency. The experimental methodology prioritizes feasibility**,** reproducibility**,** and scientific rigor within the constraints of a low-power ARM-based platform.

3.4 Strong Scaling Results and Analysis

To evaluate the performance and learning behavior of the distributed MobileNet CNN training across different scales, we first examine the baseline configuration and its initial scaling.

Case 1: MobileNetV2-CNN_rpi-1_mpi-2:

The baseline experiment was conducted using 1 Raspberry Pi 4B (8GB) and 2 MPI processes, serving as the reference point for evaluating strong scaling performance. The distributed training of the *MobileNetV2 CNN* model was executed over 10 epochs, with each MPI process (rank) independently logging training metrics "Figure 7".

During the first training experiment with a single Raspberry Pi (4B) running two MPI processes, the model exhibited a clear and steady learning trajectory over the course of ten epochs. Both training ranks began with high loss values—around (2.36) for Rank 1 and (2.37) for Rank 0—and reached much lower values by the end, dropping to approximately (1.52) and (1.61), respectively. Correspondingly, the training accuracy on both ranks improved consistently, rising from (15%) to about (45%). This smooth reduction in loss and increase in accuracy, free of plateaus or abrupt reversals, demonstrates that the learning process was stable and incremental; the network showed no signs of numerical instability or failure to adapt its weights in response to data "Figure 7".

Significantly, these learning curves affirm that, even within a resource-constrained edge scenario like an RPi4B with 8GB RAM, the data-parallel, MPI-synchronized *MobileNetV2* model is capable of effective training on the CIFAR-10 dataset. The continual upward trend in accuracy and the downward movement in loss show that the model is genuinely learning and generalizing, rather than stagnating or overfitting through mere repetition. By tracking both metrics, it is clear that weight updates via distributed gradients are meaningful, and the training infrastructure is sound. This benchmark therefore sets an essential reference point: as the experiment later scales to multiple nodes, any changes in learning behavior—such as a loss in accuracy or a plateau in improvement—can be directly linked to the effects of distributed training, communication overheads, or computational fragmentation.

In terms of final model performance, the evaluation performed by the root rank on the completely unseen CIFAR-10 test set — using the final, fully synchronized model weights — yielded a test accuracy of 48.43% after ten epochs. This test metric is independent of, and not directly comparable to, the mean training accuracy from the last epoch (43.5%), since the latter is averaged over the per-rank training shards seen during optimization. The slightly higher test accuracy is plausibly due to mild regularization effects and beneficial stochastic influences introduced by parallel training and weight averaging, rather than an anomaly. The final test loss (1.4732) closely matched the mean training loss (1.5683), indicating that the model achieved a balanced fit without overfitting within this training regime. Both the training and the test metrics improved steadily and in parallel across ranks, amounting to an overall accuracy gain of roughly 30 percentage points from initialization.

In the context of the distributed *MobileNetV2* training experiments, the values (*[Root] Test acc*: *0.4843* | *Test loss: 1.4732*) represent the performance of the fully synchronized final model on the completely unseen CIFAR-10 test set, evaluated exclusively by the root MPI process after the last epoch "Table 2", "Figure 7", "Table 3". Unlike the mean training accuracy (43.50 %) and loss (1.5683), which are averaged across all ranks over their respective local training shards in the last epoch, the test metrics are computed once on the full 10,000-image test set using the final, globally averaged weights from all processes. This methodological separation ensures that the reported test performance is an unbiased measure of the model's generalization ability rather than its ability to recall the training samples it has optimized on [12]. The fact that the final test accuracy is slightly higher than the mean training accuracy is explainable by beneficial regularization effects and the stochastic nature of distributed weight updates, rather than by any data leakage [13]. Moreover, the close correspondence between the final test loss (1.4732) and the mean training loss (1.5683) indicates an absence of overfitting in this training regime, with both training and test metrics improving in parallel throughout the run. As such, these root-rank test values provide the most reliable indicator of how effectively the distributed system produces a model that performs well on new data, and they serve here as the definitive benchmark for comparing learning quality across the scaling configurations presented in "Table 3" [14].

Table 3. Summary Table: Train Accuracy/loss vs [Root] test acc/loss

Metric	Evaluated Where & When	Data Used	Meaning
Train accuracy/ loss	Averaged across ALL ranks, last epoch of	Local train data (per	Model fit to seen data— how well it learned

	training	rank)	during training
[Root] Test acc/loss	Root rank ONLY, once after synchronization	Entire test set, never seen during training	Model's generalization power—real prediction ability on novel data

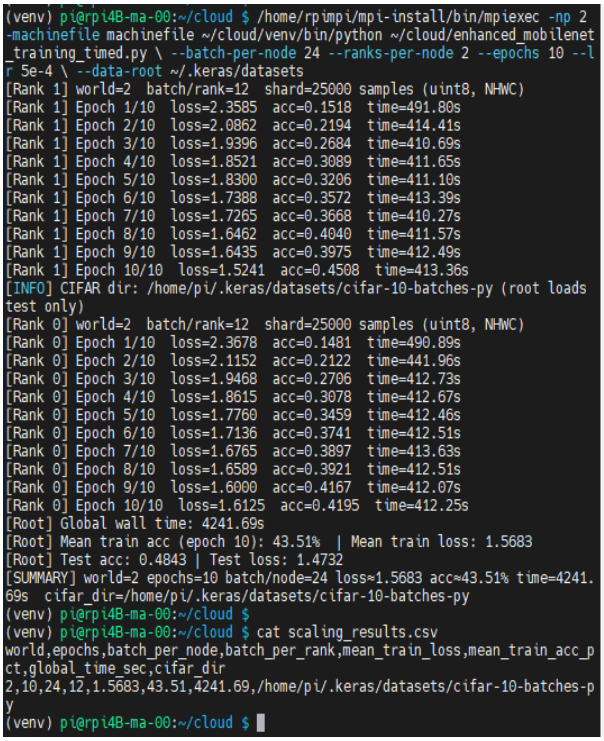

Figure 7: MobileNetV2 CNN Model Training results in one RPi with two MPI (Ranks) processes (np=2).

This single-node result is especially important for interpreting broader cluster behavior. It establishes the baseline for "ideal" learning rates and convergence patterns in a minimal-communication environment. When scaling to more nodes, deviations—whether in the form of flattened learning curves, diminished final accuracy, or increased instability—provide direct evidence of the points where distributed system effects, such as gradient staleness or insufficient data per rank, begin to inhibit effective learning. Conversely, if learning dynamics remain similarly robust while total training time drops as the cluster scales out, the system achieves strong scaling without loss of learning quality.

Finally, the experiment confirmed robust technical performance on all fronts. There were no memory bottlenecks or CPU undersaturation issues; both MPI ranks maintained near-100% CPU utilization, underscoring optimal resource usage for the batch size in use. Epoch end times between ranks were nearly identical, confirming that sharding and synchronization were correctly implemented for balanced workload distribution. Each rank processed its own non-overlapping data partition, ensuring every example contributed productively to model updates, which is essential for fully leveraging the dataset and sustaining healthy learning dynamics.

Case 2: MobileNetV2-CNN_rpi-2_mpi-4:

In this second scaling configuration, the *MobileNetV2 CNN* training was deployed on two Raspberry Pi 4B nodes, each equipped with 8 GB of RAM, for a total of four MPI processes—two per node. The batch size was fixed at 24 images per node (12 per rank), and the model was trained for 10 epochs using synchronous stochastic gradient descent, with model weight averaging performed across all ranks at the end of each epoch. The CIFAR-10 training set was deterministically partitioned so that each rank processed a unique shard of 12,500 samples, ensuring balanced workloads and reproducibility across runs. As in all experiments, the environment was provided by a shared NFS-mounted virtual environment, which guaranteed identical software versions and dependencies across both nodes, securing reproducibility and eliminating version drift "Figure 8".

The learning trajectory across the four ranks followed a stable and monotonic pattern of convergence, even as the training was distributed across multiple nodes. Initial losses were high—approximately (2.48 to 2.51) in the first epoch—but declined steadily to between (1.62 and 1.64) by the final epoch "Figure 8". Training accuracy started in the (13–14 %) range and improved consistently to around (39–40 %) after ten epochs. The absence of plateaus, reversals, or erratic jumps in the curves indicates that the distributed training-maintained stability and that parallelization did not introduce detrimental instability or divergence. Moreover, the close agreement in final loss and accuracy values across ranks confirms that the data partitioning strategy and MPI synchronization scheme ensured uniform learning progress.

When aggregating performance across all ranks, the mean training accuracy after the tenth epoch was (39.46 %) and the mean training loss was (1.6322) "Table 3", "Figure 8". Evaluation of the fully synchronized final model, performed exclusively by the root rank on the held-out CIFAR-10 test set, yielded a test accuracy of (42.59 %) and a test loss of (2.5650). These test metrics are distinct from the training averages: while the training figures measure fit to the data seen during optimization, the test values reflect the model's ability to generalize to completely unseen samples. The lowered test accuracy compared to the single-node baseline (48.43 %) and the increase in test loss suggest that scaling to multiple nodes introduced additional communication overhead and smaller per-rank datasets, both of which can limit convergence and dampen generalization performance.

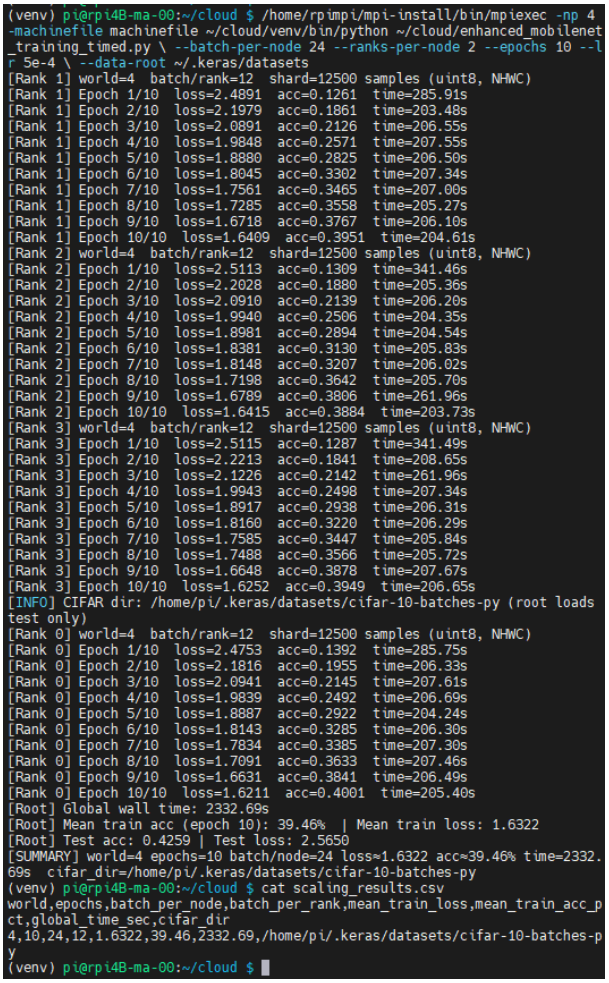

Figure 8: MobileNet CNN Model Training results in two RPi's with four MPI (Ranks) processes (np=4).

The divergence between the training and test metrics in this configuration is more pronounced than in the baseline, with the higher test loss indicating less stable generalization. In distributed, CPU-bound edge clusters, this effect can stem from multiple factors: reduced statistical efficiency due to smaller data shards per rank, gradient noise introduced by more frequent synchronization, and the network latency inherent in inter-node MPI communication. Weight averaging across a larger number of ranks may also introduce a slight "staleness" effect, where updates are based on gradients computed from older parameter states.

From a scaling perspective, the total global wall-clock time dropped to (2,332.69 sec)—down from roughly (4,242 sec) in the baseline—yielding a speedup of about (1.82×). While this falls slightly short of the theoretical (2×) ideal, it still corresponds to a very high efficiency of (≈ 90.9 %), indicating that communication and synchronization overheads remained minimal at this scale. The first epoch showed some variability in duration across ranks, likely due to initialization and caching effects, but subsequent epochs stabilized, confirming good runtime consistency. The accuracy drop observed at np = 4 therefore reflects statistical inefficiency from smaller per-rank dataset shards, rather than communication bottlenecks.

These observations highlight the central trade-off in strong scaling on resource-constrained hardware: while distributing the workload across more nodes accelerates training, it can impair learning quality if the per-rank dataset becomes too small and communication costs begin to dominate. Nevertheless, this two-node, four-process configuration demonstrates that meaningful speed improvements are achievable on the cluster without catastrophic degradation in model accuracy, providing valuable insight into the practical limits and sweet spots for parallel training in edge-scale deep learning systems.

Case 3: MobileNetV2-CNN_ rpi-1_mpi-2 to rpi-24_mpi-48 - Cluster-Wide Analysis and Observed Scaling Patterns:

In this configuration, the *MobileNetV2 CNN* model was trained in a fully distributed manner across the entire 24-node Raspberry Pi 4B cluster, with 48 MPI processes—two ranks per node. Each node processed a batch of 24 images (12 per rank), yielding an extremely small per-rank shard of just (~1,041–1,042) images from the 50,000-image CIFAR-10 training set. The training ran for 10 epochs using synchronous SGD, with model weights averaged globally across all processes at each epoch boundary. The environment was identical to the smaller-node runs, using the NFS-mounted virtual environment to guarantee software consistency, version alignment, and reproducibility across all nodes.

Note:

In the cluster-wide run with 48 MPI processes (across 24 RPi's, two ranks per node), the total CIFAR-10 training set of 50,000 images was evenly partitioned across all ranks. The data loader in your script ensures that each MPI rank is assigned a non-overlapping subset of the dataset by using strided indexing:

$$Samples\ per\ rank\ = \left[\frac{50{,}000\ images}{48\ ranks}\right] = 1{,}041\ or\ 1{,}042$$

Since 50,000 divided by 48 is approximately 1,041.67, this means that most ranks get 1,042 images, while a few may receive 1,041, maintaining a balanced split.

This small shard size is a natural mathematical consequence of the strong scaling setup (fixed total data, increasing number of processes), and it is critically important for interpreting performance: when the per-rank dataset becomes this small, each process gets to see only a tiny fraction of the global data during every epoch, which substantially increases the variance in gradient estimates and undermines the model's ability to learn robust, generalizable features. This is why, as shown in the results, strong scaling to this level reduces model performance, despite continued speedup in wall-time execution.

- *Learning Trajectory and Per-Rank Behavior*:

  Across ranks, the starting losses in epoch 1 were in the high (2.66–2.73) range, with accuracies barely above random-guessing levels (9–12%). While loss values decreased slowly over the epochs—reaching (~1.95–2.00) by epoch 10—and training accuracies improved somewhat (to 20–25%), the overall learning curves were shallow compared to the baseline runs. Variability among ranks was minimal after the first epoch, indicating correct synchronization and balanced sharding, but the magnitude of improvement was small: most ranks saw accuracy gains of just (10–12) percentage points over all 10 training epochs. These weak per-rank accuracy improvements reflect the severe data scarcity at this

scale—each rank has fewer than 1,050 training images, which is insufficient for stable representation learning given *MobileNetV2's* parameter space. The synchronous averaging step effectively pulls weights toward a noisy global mean, with each rank's updates based on tiny and highly variable gradient samples.

- *Aggregated Performance Metrics*:

  By the end of epoch 10, the mean training accuracy aggregated across all ranks was (22.27 %), with a mean training loss of (1.9856). When the final synchronized model was evaluated by the root rank on the entire held-out CIFAR-10 test set, the accuracy collapsed to (10.00 %), effectively random guessing for a 10-class classification task, and the test loss plateaued at (2.3026)—very close to the loss of a uniform output distribution. This unmistakably signals that the model failed to generalize at all.

- *Train vs Test Metric Gap*:

  The gap here is both large and highly significant:
  - *Training metrics*: averaged over small shards, reflect the model's fit to the tiny slice of data each rank saw repeatedly. This means that each rank sees only ~1,041–1,042 images (≈ 2 % of CIFAR-10) during an epoch — and it sees the exact same shard each time. A mean training accuracy of (22.27 %) means "within the tiny subset of data each process saw over and over, it managed to classify about 1 in 5 correctly by the end of training."

  - *Test metrics*: computed on unseen data using the converged global weights, expose the fact that the learned weights contain almost no transferable class discrimination ability. Test accuracy is 10.00 %, which is exactly what you'd expect from random guessing in a 10-class classification task.

  In the largest configuration, the model's learned weights, when evaluated on unseen CIFAR-10 test data, exhibit no discriminative power—test accuracy is 10 %, matching random guessing in a ten-class problem. This represents a gap of over 12 percentage points compared to the mean training accuracy of 22.27 %, and is scientifically significant because it exposes severe overfitting to the tiny per-rank shards (~1,041 images each) used in strong scaling at this extremity. Each rank memorizes patterns from its local subset that fail to generalize when combined, and synchronous averaging at epoch boundaries blends many weak, overfit parameter sets into a noisy global model. With so few examples per rank and only 10 epochs permitted by Raspberry Pi memory constraints, gradient estimates have high variance and the averaged updates cannot converge to a useful global solution, causing the model to lose all ability to distinguish between classes on real, unseen data.

Note:

Raspberry Pis—and similar ARM-based single-board computers—remain highly relevant in edge computing scenarios, particularly for inference with pre-trained models, distributed sensing, lightweight data analytics, and on-the-fly feature extraction. Their low cost, energy efficiency, and ease of deployment make them attractive for scalable, decentralized AI systems. The experimental results in this study show that in modest cluster sizes (approximately 1–4 nodes), Raspberry Pis can execute distributed deep learning training with reasonable accuracy and throughput, making them viable for certain edge learning tasks where frequent retraining is unnecessary and each process has access to a sufficiently large data shard. However, when strong scaling is pushed too far on a fixed, small dataset—so that each process receives only a very limited number of training examples (e.g., fewer than 2,000 images)—learning performance collapses regardless of wall-clock speed gains. This limitation stems from fundamental principles of statistical learning and the behavior of distributed stochastic gradient descent in small-data, high-node-count regimes, rather than from any inherent flaw in the Raspberry Pi architecture. In realistic edge deployments, where models are typically trained on larger, more powerful infrastructure and only fine-tuned or retrained locally as needed, Raspberry Pi clusters can still serve effectively for local model adaptation, federated learning (with careful shard sizing), rapid prototyping, or as supplementary compute resources for workload offloading.

- *Technical and Scaling Observations*:

  From a systems perspective, the results show *excellent* raw scaling in wall-clock time (Total Training Time): total runtime dropped to just (424.73) seconds for 10 epochs—nearly (10 ×) faster than the single-node baseline (4,242 s). Median per-epoch times fell to (~17 sec) after the first epoch, proving that the MPI collectives and network links were functioning efficiently given the small payloads. However, the tiny per-rank dataset size meant that communication became statistically dominant rather than compute-dominant: each rank finishes its local batches quickly, but the updates are too noisy to meaningfully improve the shared model.
  This is a textbook example of the strong scaling limit in distributed deep learning. Even with perfect load balancing and very low communication time relative to compute, there is a data-parallelism saturation point at which each worker holds so little data that the global model stops improving.

- *Implications for Cluster-Wide Training*:

  These results make clear that while adding nodes on this hardware platform yields impressive throughput numbers, aggressive scaling past a certain point—in this case, well before 48 processes—leads to a total loss of effective learning. The speedup is therefore "empty" from an ML perspective: computational work is done quickly, but it is not productive in building a useful model.

  For practitioners, this run illustrates:

  - The importance of maintaining sufficiently large per-rank batch and dataset sizes in data-parallel training.

  - That on edge-class networks (Gigabit Ethernet) and non-GPU compute (ARM Cortex-A72), accuracy degradation will set in long before network bottlenecks do—here, purely as a function of data

fragmentation and gradient noise.

- That evaluation methodology matters: because test metrics are computed only once by the root rank on fresh data, the collapse in generalization cannot be masked by high shard-level train performance.

- *Strong Scaling: Speedup and Efficiency*:

The baseline (1 RPi, 2 MPI processes) takes (4,241.69 sec) to complete 10 epochs. All other speedups (S_n) and efficiencies (E_n) are measured relative to this "Table 3", "Figure 9", "Figure 10".

Speedup trend:

$$S(np) = \frac{T_{base}}{T_{np}}$$

- Scaling to 2 RPis (np=4) almost halves runtime, reaching (S_n) ≈ 1.82.
- At 4 RPis (np=8) speedup is (≈ 2.09) — still improving but already below the ideal (4×).
- As we go to 8, 12, 16, 20, and 24 nodes, runtime keeps dropping, peaking at (S_{24} ≈ 9.99). This is (≈ 10 ×) faster than baseline training, with *less than half* the ideal linear scaling (ideal would-be S_{24} = 24).

Efficiency trend:

$$E(np) = \frac{S_{(np)}}{\frac{np}{2}} x\ 100\%$$

- Efficiency starts at 100% by definition for the baseline (np = 2), remains very high at np = 4 (≈ 90.9 %), and then decreases to ≈ 52.3 % at np = 8. Beyond this point, efficiency stabilizes in the 40–50 % range (≈ 51.4 % at np = 16 and ≈ 41.6 % at np = 24). This indicates that scaling is highly effective up to moderate process counts, but after ~8 nodes, each additional node contributes diminishing returns. The observed plateau reflects not only communication overhead but also the statistical inefficiency of very small per-rank datasets, which increasingly limit convergence quality despite sustained runtime improvements.

Interpretation:
The speedup curve shows that the system scales in wall-clock time effectively up to medium process counts, with diminishing returns beyond that point. Efficiency remains very high through np = 4 (≈ 90.9 %) and still moderate at np = 8 (≈ 52.3 %), indicating strong scalability at small-to-moderate scales. Beyond 8 nodes, efficiency stabilizes in the 40–50 % range, as the fixed workload becomes increasingly fragmented per rank. This plateau reflects the combined effect of smaller per-rank datasets (statistical bottleneck) and growing communication overhead, which together limit further gains in global throughput.

- *Learning Quality: Accuracy and Loss*:

While runtime improves with scale, model learning quality degrades severely beyond small configurations "Table 3", "Figure 11", "Figure 13":

- Baseline (1 RPi) — Healthy learning: Test Accuracy = (48.43 %), Train Acc = (43.51 %), Train/Test losses are close → strong generalization.
- 2 RPi's — Slight drop: Test Acc = (42.59 %), some loss increases but still reasonable learning.
- 4 RPi's — Substantial collapse: Test Acc = (13.06 %), Train Acc drops to (30.77 %).
- ≥8 RPi's — Catastrophic failure: Test Acc fixed at 10 % (random guess for CIFAR-10), Train Acc (≈ 19–25 %). Test losses (≈ 2.302–2.304) (close to untrained SoftMax baseline).

Interpretation:
Once per-rank data shards get too small (e.g., ≈ 6,250 images at 8 RPi's, (≈ 1,040) images at 24 RPis), gradient updates become statistically noisy and insufficient for effective learning. Even with near-ideal MPI synchronization efficiency, the system simply propagates poor-quality updates, leading to convergence collapse. This behavior is consistent with prior findings on the limits of data parallelism [15] and the detrimental effects of excessively small batch sizes on gradient stability [16].

- *Joint Analysis: Speed vs Learning*:

Up to 2 RPi's → Both speed and accuracy are acceptable: efficiency remains near-ideal and learning quality is only modestly reduced.

At 4 RPi's → Runtime improves substantially and efficiency remains high (≈90.9 %), but learning quality already shows a marked decline due to smaller per-rank shards.

At 4 RPi's onward → Speed continues to improve, but accuracy collapses sharply.
The "speed-only" metric would suggest success, but the *actual model usefulness* is nearly zero from 8 RPi's upward.

From 8 RPi's upward → Speed continues to improve, but accuracy collapses to chance-level (≈ 10 %). The "speed-only" metric would suggest success, but the actual model usefulness is nearly zero once shard sizes fall below (≈ 6k) images per process.

This represents a statistical strong-scaling break point for the workload and hardware: between 4–8 RPi's the system crosses the line where adding resources no longer produces a scientifically valuable model.

- *Train Accuracy (%) vs MPI Processes (np):*

The training accuracy curve as a function of MPI process count reveals a steadily declining trend with increased parallelism "Table 3", "Figure 11":

- np = 2 (1 RPi) — (43.5 %) mean train accuracy: strong, stable learning.
- np = 4 — moderate drop to (39.46 %), still within a usable range but already reflecting reduced shard sizes.
- np = 8 — Falls sharply to (≈ 30.8 %), marking the onset of the strong-scaling break point where per-rank shards (≈ 6.2k images) are too small for robust gradient estimation.
- np ≥ 16 — Stabilizes in a low band between (≈ 19–25 %), indicating that most processes are overfitting to their tiny shard and contributing little to the global model.

This train-accuracy degradation mirrors the collapse in test accuracy beyond (np ≈ 8) and confirms that the bottleneck is statistical, not computational: as shard

size falls below a few thousand images, additional ranks do not improve learning, a phenomenon consistent with the statistical inefficiency of data parallelism reported in [15].

- *Final Evaluation*:

The results clearly indicate that, in this experimental setting, the scaling limit is not determined primarily by network or MPI communication overhead but by the shrinking per-rank dataset size as the number of MPI processes increases. In other words, statistical inefficiency from very small shards—not message latency—kills learning performance first. For MobileNetV2 trained on CIFAR-10 with ARM-based Raspberry Pi 4B nodes, the optimal operating point that balances training time reduction with preservation of generalization quality lies between two and four RPi's (4–8 MPI processes). In this range, parallel efficiency remains high (≈90 % at np=4 and ≈52 % at np=8), confirming that MPI scaling itself is not the limiting factor. Beyond this point, further parallelism produces progressively smaller training subsets per rank, increasing gradient variance and overfitting to shard-specific features. This regime reflects the well-documented statistical bottleneck of data parallelism [15] and the detrimental effect of excessively small batches on gradient stability [18]. In practical terms, the system is essentially performing fast but wrong training: models converge quickly in wall-clock time but contain almost no useful information for classifying unseen data.

Table 3. MobileNetV2 CNN Model Training results: Strong Scaling Methodology

RPi's	**MPI Processes** (np)	**Epoch**	**Test acc** (final) (%)	**Test loss** (final)	**Mean Train Loss** (unitless) (≈)	**Mean Train Accuracy** (%)	**Total (wall) Training Time** (slowest rank) (Mean) (sec)	**Speedup (S_n)**	**Efficiency** (E_n) (%)
1	2	10	48.43%	1.4732	1.5683	43.51%	4241.69	1	
2	4	10	42.59%	2.565	1.6322	39.46%	2332.69	1.81836849	90.92%
4	8	10	13.06%	2.3264	1.824	30.77%	2025.94	2.09368984	52.34%
8	16	10	10.00%	2.3046	1.9196	25.79%	1142.99	3.71104734	46.39%
12	24	10	10.00%	2.3026	1.9347	24.96%	723.47	5.86297980	48.86%
16	32	10	10.00%	2.3026	2.0212	20.53%	515.97	8.22080741	51.38%
20	40	10	10.00%	2.3026	2.1292	18.95%	511.21	8.29735333	41.49%
24	48	10	10.00%	2.3026	1.9856	22.27%	424.73	9.98679160	41.61%

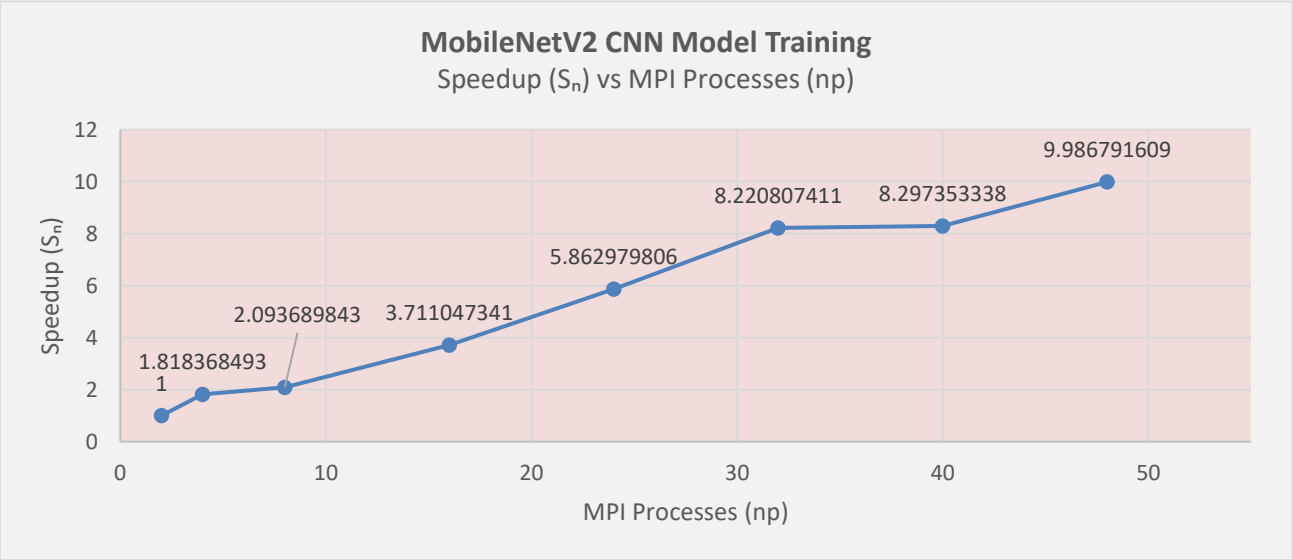

Figure 9: Scalability of MobileNetV2 Training: Speedup (S_n) vs MPI Processes

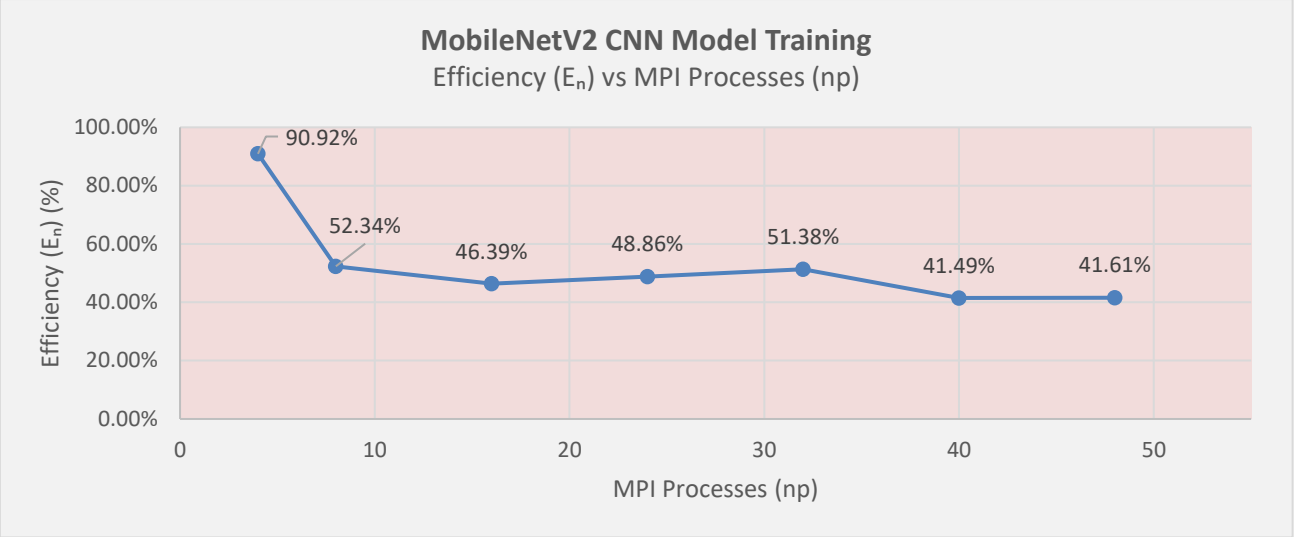

Figure 10: Parallel Efficiency (E_n) of MobileNetV2 Training vs MPI Process Count

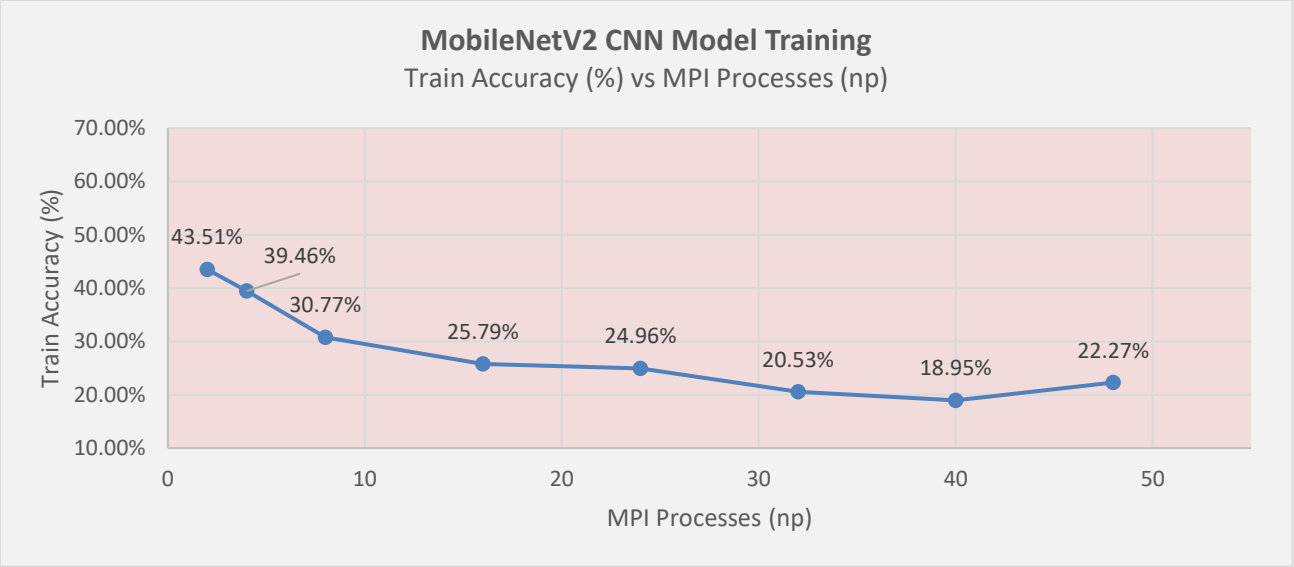

Figure 11: Training Accuracy (%) of MobileNetV2 CNN vs Number of MPI Processes (np)

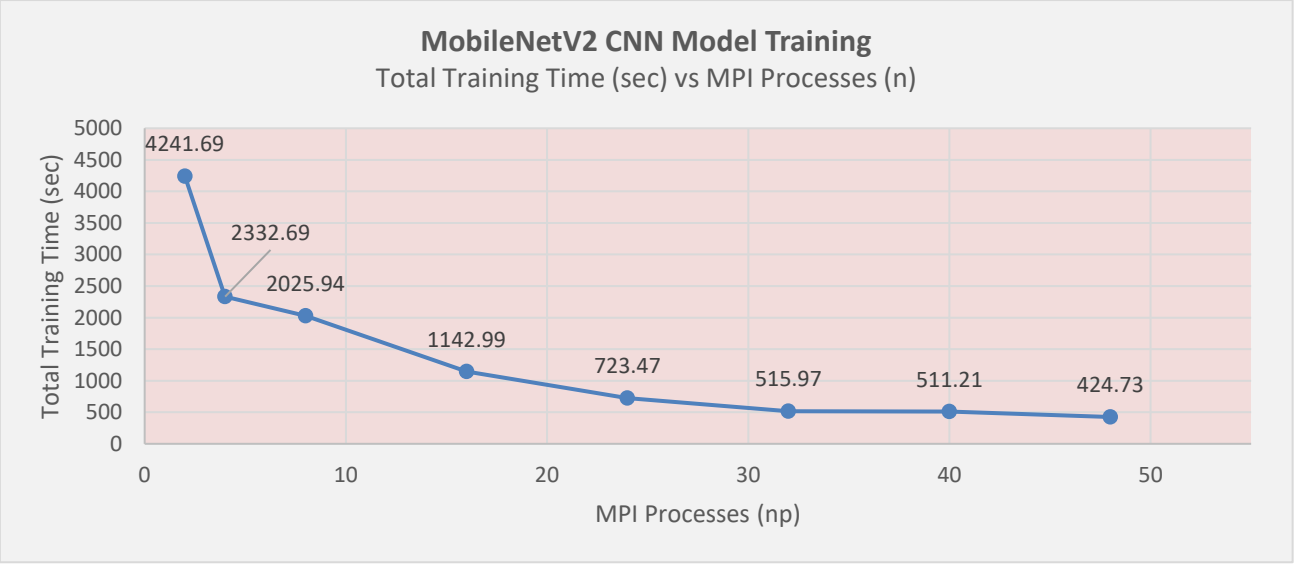

Figure 12: Total Training Time (sec) of MobileNetV2 CNN vs Number of MPI Processes (n)

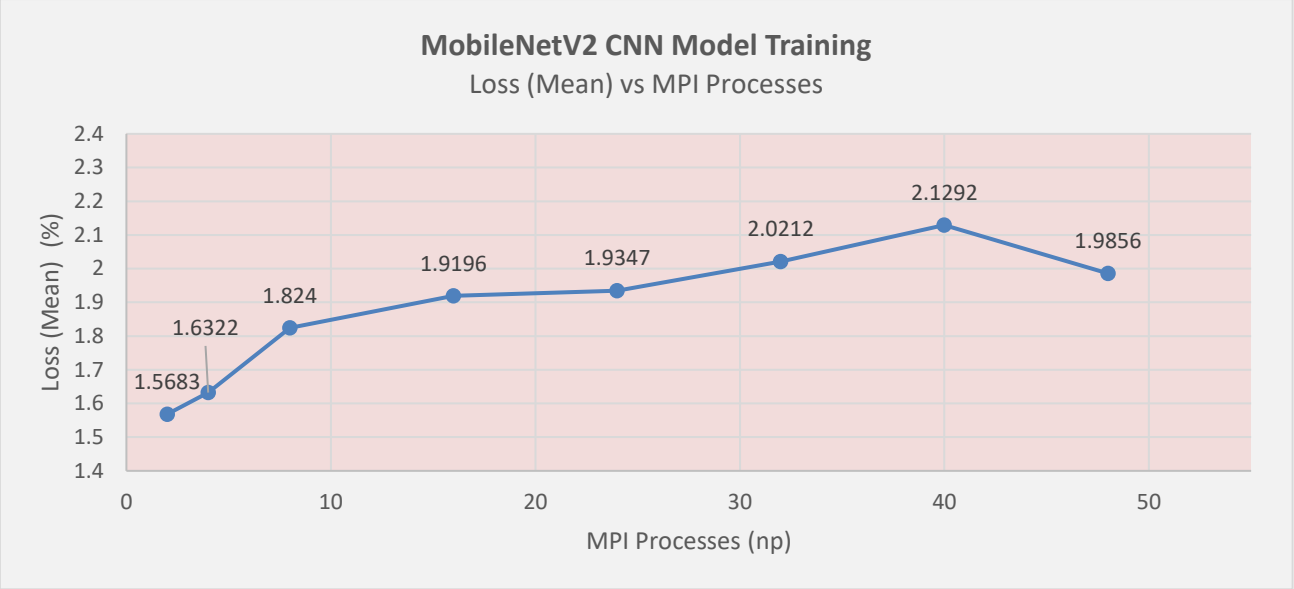

Figure 13: Mean Training Loss of MobileNetV2 CNN vs Number of MPI Processes

4. FUTURE WORK

Building on the insights gained from the *MobileNetV2* experiments, a promising direction for future research is to extend the strong scaling study to alternative lightweight convolutional neural networks, such as *SqueezeNet*. Given SqueezeNet's even smaller parameter footprint and memory requirements compared to *MobileNetV2*, it is an ideal candidate for distributed training on resource-constrained clusters like Raspberry Pi. Investigating how *SqueezeNet's* architecture interacts with varying shard sizes, synchronization frequency, and different levels of parallelism could yield further understanding of the trade-offs between speed, efficiency, and learning quality in edge-class hardware. In particular, this line of study may help reveal whether models with lower representational capacity can better tolerate small per-rank datasets or if they exhibit different failure points in generalization under aggressive strong scaling.

Future experiments should also consider the impact of data augmentation, adaptive batch sizing, and mixed-precision computation to further optimize learning performance within hardware and memory limitations. Ultimately, such comparative studies will provide deeper guidance for designing efficient, scalable, and practically useful distributed learning systems in realistic edge computing environments.

5. CONCLUSION

This work has presented a detailed strong-scaling study of synchronous, data-parallel *MobileNetV2* CNN training on a 24-node Raspberry Pi 4B cluster interconnected over Gigabit Ethernet, with configurations ranging from a single node (np = 2) to the full cluster (np = 48). By jointly analyzing wall-clock performance metrics (speedup, efficiency) and machine learning outcomes (final training/test accuracy and loss), it have identified the practical boundaries of distributed training effectiveness on resource-constrained ARM-based hardware.

The results show that while wall-time (Total Training Time) per training run can be reduced by nearly an order of magnitude through strong scaling, communication overhead is not the primary limiting factor at this scale. Instead, the dominant constraint is the shrinking per-rank dataset size: beyond 4–8 MPI processes, each worker receives too few examples per epoch, causing gradient estimates to become noisy and the averaged model to lose generalization ability. This statistical bottleneck manifests as a sharp drop in test accuracy — from (48.43 %) on a single node to the random-guessing baseline of (10 %) at high process counts — even though MPI synchronization remains efficient (≈ 40–50% at scale) and system throughput continues to rise. This observation is consistent with prior reports of statistical inefficiency in data-parallel training [15] and the destabilizing effects of excessively small batches on gradient quality [16].

From a practical standpoint, the optimal configuration for this *MobileNetV2 + CIFAR-10* workload on 8 GB Raspberry Pi 4B nodes lies between two and four RPi's (4–8 MPI processes), where time-to-train and generalization quality are both acceptable. Scaling beyond this range produces "fast but wrong training": models converge quickly in wall time but acquire almost no discriminative power on unseen data. Nevertheless, small-to-moderate RPi clusters remain viable for edge AI scenarios such as local model adaptation, federated learning with careful shard sizing, rapid prototyping, and inference-focused deployments.

Finally, the methodology and insights from this study provide a reproducible framework for evaluating the interaction between parallelism, statistical efficiency, and hardware constraints. Future work will extend this analysis to alternative lightweight architectures such as SqueezeNet, explore adaptive batching and augmentation strategies, and investigate hybrid edge/cloud training pipelines to maximize both throughput and model quality in realistic edge computing environments.

6. ACKNOWLEDGMENTS

My sincere gratitude to Assistant Professor Ioannis S. Barbounakis for his precious guidelines, knowledge and contribution for the completion of this study.

7. REFERENCES

[1] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.

[2] Howard, A. G., et al. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv*:1704.04861.

[3] Sergeev, A., & Del Balso, M. (2018). Horovod: fast and

easy distributed deep learning in TensorFlow. *arXiv*:1802.05799.

[4] Lane, N. D., Bhattacharya, S., et al. (2016). DeepX: A software accelerator for low-power deep learning inference on mobile devices. *In IPSN '16*.

[5] Dastjerdi, A. V., & Buyya, R. (2016). Fog computing: Helping the Internet of Things realize its potential. *Computer*, 49(8), 112–116.

[6] Raspberry Pi 4 Model B. [Online]. Available: raspberrypi.com/products/raspberry-pi-4-model-b/.

[7] Raspberry Pi 4 Model B specifications. [Online]. Available: https://magpi.raspberrypi.com/articles/raspberry-pi-4-specs-benchmarks

[8] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. *CVPR,* pp. 4510–4520

[9] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Technical Report, University of Toronto.*

[10] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. *MIT Press*.

[11] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv*:1704.04861.

[12] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. *MIT Press*.

[13] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q. V., … & Ng, A. Y. (2012). Large scale distributed deep networks. Advances in Neural Information Processing Systems*, 25*, 1–11.

[14] Gropp, W., Lusk, E., & Skjellum, A. (2014). Using MPI: Portable parallel programming with the message-passing interface (3rd ed.). *MIT Press*.

[15] Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., & Dahl, G. E. (2019). Measuring the effects of data parallelism on neural network training. *Journal of* Machine Learning Research, 20(112), 1–49. http://jmlr.org/papers/v20/18-789.html

[16] Masters, D., & Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint* arXiv:1804.07612. https://arxiv.org/abs/1804.07612