SoCFlow: Efficient and Scalable DNN Training on SoC-Clustered Edge Servers

Daliang Xu[♦*], Mengwei Xu[◇*], Chiheng Lou[♦], Li Zhang[◇], Gang Huang[♦▲], Xin Jin[♦], Xuanzhe Liu[♦]
[♦]Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education; School of Computer Science, Peking University, Beijing, China
[◇]State Key Laboratory of Networking and Switching Technology, China
[▲]National Key Laboratory of Data Space Technology and System, China
{xudaliang,hg,xinjinpku,liuxuanzhe}@pku.edu.cn,louchiheng23@stu.pku.edu.cn
{mwx,li.zhang}@bupt.edu.cn

Abstract

SoC-Cluster, a novel server architecture composed of massive mobile system-on-chips (SoCs), is gaining popularity in industrial edge computing due to its energy efficiency and compatibility with existing mobile applications. However, we observe that the deployed SoC-Cluster servers are not fully utilized, because the hosted workloads are mostly user-triggered and have significant tidal phenomena. To harvest the free cycles, we propose to co-locate deep learning tasks on them.

We present `SoCFlow`, the first framework that can efficiently train deep learning models on SoC-Cluster. To deal with the intrinsic inadequacy of commercial SoC-Cluster servers, `SoCFlow` incorporates two novel techniques: (1) the group-wise parallelism with delayed aggregation that can train deep learning models fast and scalably without being influenced by the network bottleneck; (2) the data-parallel mixed-precision training algorithm that can fully unleash the heterogeneous processors' capability of mobile SoCs. We have fully implemented `SoCFlow` and demonstrated its effectiveness through extensive experiments. The experiments show that `SoCFlow` significantly and consistently outperforms all baselines regarding the training speed while preserving the convergence accuracy, e.g., 1.6×–740× convergence speedup with 32 SoCs. Compared to commodity GPU (NVIDIA V100) under the same power budget, `SoCFlow` achieves comparable training speed but reduces energy consumption by 2.31×–10.23× with the same convergence accuracy.

CCS Concepts: • **Hardware → Emerging architectures**; • **Human-centered computing → Ubiquitous and mobile computing systems and tools**; • **Computing methodologies → Distributed artificial intelligence**.

Keywords: SoC-Cluster, distributed machine learning, mixed precision training

ACM Reference Format:
Daliang Xu[♦*], Mengwei Xu[◇*], Chiheng Lou[♦], Li Zhang[◇], Gang Huang[♦▲], Xin Jin[♦], Xuanzhe Liu[♦]. 2024. SoCFlow: Efficient and Scalable DNN Training on SoC-Clustered Edge Servers. In *28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3617232.3624847

1 Introduction

SoC-Cluster, a new form of server that comprises massive mobile system-on-chips (SoCs), is emerging in the edge computing [14, 110]. Compared to traditional servers deployed in cloud/edge datacenters, SoC-Cluster servers deliver denser computing capacity (e.g., 60x Snapdragon 865, thus 480x physical cores in a 2U rack, details in §2.1), higher energy efficiency, and more importantly, the ability to run native mobile applications without any modification. As a result, millions of such SoC-Clusters have been deployed in edge clouds to serve mobile applications, such as cloud gaming [4–7, 13] and live streaming [12].

However, as shown in §2, the traces that we collected from thousands of SoC-Cluster servers in a real-world deployment reveal that they are severely under-utilized, e.g., the average CPU usage of more than 95% SoCs is under 20%. A primary reason for this is that the applications hosted on those servers are mostly triggered by user interaction activities, resulting in significant fluctuations in resource usage (e.g., tidal phenomenon). For instance, the CPU usage of the SoCs is approximately 50× lower at midnight compared to peak hours.

∗: Equal contributions.

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0372-0/24/04…$15.00
https://doi.org/10.1145/3617232.3624847

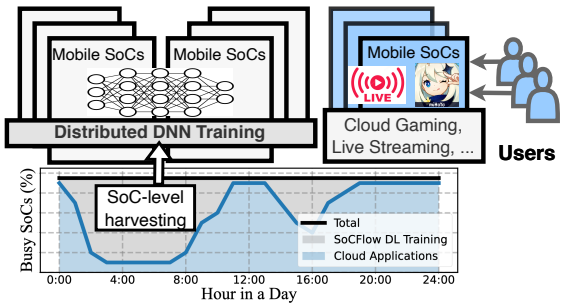


Figure 1. SoCFlow improves SoC-Cluster resource utilization by interleaving DNN training and existing user-triggered workloads.

To harvest such free cycles, this paper proposes to perform deep learning (DL) training tasks on idle SoCs, as shown in Figure 1. We use DL training to fill the usage gap of SoC-Cluster for three reasons. (i) Training DNN models on edge clouds is increasingly promising and pervasive [9, 24, 40, 57, 95, 101, 103]. Edge clouds are where mobile user data can converge. Compared to training models on a centralized data center, it cuts down the network route between data and training infrastructure, thus reducing the network pressure of the backbone network. Furthermore, training on edge clouds also mitigates privacy concerns by early consuming the data. It also allows model training to be personalized geographically, e.g., item recommendation tasks [39, 66, 91]. (ii) DL training is a relatively predictable, delay-tolerable workload that complements existing workloads on SoC-Cluster. For instance, an input method prediction model [103] could be periodically (e.g., per month) updated on each region's edge cloud within one midnight and dispatched to mobile clients the next morning. (iii) In recent years, significant research efforts have demonstrated that mobile SoCs with on-chip accelerators (e.g., DSPs and NPUs) can achieve remarkable capabilities in executing deep learning tasks [37, 47, 51, 56, 60, 61, 89]. Meanwhile, algorithm foundations for low-precision training that fit into on-chip accelerator data format have been established [94, 114, 123] for important machine learning tasks, particularly those relatively small models and simple computer vision tasks that SoCFlow targets. Consequently, this research landscape has made on-SoC training a practical endeavor [101].

Unfortunately, our initial experiments in §2.3 show that a single SoC is insufficient to train a medium-sized model within a reasonable time frame. For instance, training VGG-11 [86] on CIFAR-10 takes 29.1 hours on Snapdragon 865 CPU, or 7.5 hours on NPU in INT8 data format with 2.7% accuracy loss. This prolonged training time not only delays the deployment of the updated model to users but also complicates the software design as training needs to span multiple idle periods.

Intuitively, like model training on cloud servers, we can orchestrate multiple SoCs to accelerate DNN training, i.e., *distributed training*. While this technique has been extensively explored in datacenters [54, 65, 76, 83–85, 87, 92, 109], our experimental in §2.3 reveal two distinct challenges specific to the SoC-Cluster.

• *Scarce network bandwidth.* While datacenter networks offer up to 100Gbps bandwidth [76], the cross-SoC network in a typical SoC-Cluster server is restricted to only 1Gbps. To make matters worse, such scarce network capacity is shared by tens of SoCs. This is attributed to the underlying SoC board design, where all cross-SoC network traffic goes through a single centralized network switch. Indeed, it is enough to support applications like cloud gaming, yet easily bottlenecks with intensive cross-SoC network traffic. As a result, we find existing network topologies deployed in datacenters for distributed training, such as parameter server [54] or Ring-AllReduce [85], are unable to scale the training speed with more participate SoCs.

• *Heterogeneous processors with mixed data formats.* Domain-specific accelerators are commonly used to speed up DL workloads. Mobile SoCs are heterogeneous as well, but: (i) mobile GPUs are demonstrated to be inefficient for training tasks [25, 33, 111]; (ii) mobile NPUs operate on low-precision format, e.g., INT8, and speed up training at the price of accuracy degradation [101]. Consequently, we need a mixed-precision training algorithm and an aggregation scheme across mobile CPUs and NPUs, well to balance the training speed and DNN convergence accuracy.

In this paper, we present SoCFlow, the first framework that can efficiently train DNN models on SoC-Cluster servers. The main design goal of SoCFlow is to scale the training speed with the number of participant SoCs, particularly for small-to-medium-sized models that edge cloud needs. We design two novel techniques to address the above challenges.

Group-wise parallelism with delayed aggregation (§3.1). Inspired by the prior efforts on communication-efficient distributed training [28, 74, 97, 107, 115] and federated learning protocols [24], SoCFlow employs a hierarchical network topology to mitigate the network bottleneck. Specifically, SoCFlow divides the SoCs into *groups*: within a group, the SoCs form a Ring-AllReduce topology and exchange their weight updates frequently (e.g., per batch); across groups, the weights are aggregated in a delayed manner (e.g., per epoch) akin to federated learning. To fully unleash the SoC parallelism, SoCFlow incorporates three steps in determining the concrete topology and runtime strategy: (1) determining a proper group size through a novel utility function that jointly considers the training speed and cross-group data distribution gap; (2) judiciously mapping the logical hierarchical topology into the concrete physical SoC-Cluster architecture with an integrity-greedy mapping algorithm, seeking to minimize the communication overhead; and (3)

carefully ordering the group-wise communication at runtime to reduce network contention.

Data-parallel mixed-precision training (§3.2). SoCFlow leverages mobile CPU and NPU for data-parallel training with weights/gradients in FP32 and INT8 formats, respectively. The mixed-precision aggregation is performed on the chip before cross-SoC synchronization. SoCFlow further introduces two key metrics: one that estimates the accuracy gap between the logits from the CPU and NPU; and another that measures the compute power gap between the CPU and NPU. Utilizing these metrics, SoCFlow judiciously partitions the per-batch training data across CPU/NPU to optimize the training speed and to mitigate the precision loss of INT8-based training by offloading part of the training to the CPU with FP32 format.

We have fully implemented SoCFlow on top of MNN [53], the state-of-the-art training library on mobile SoCs. SoCFlow supports models exported from TensorFlow [18] and PyTorch [65]. We have also comprehensively evaluated the system on a commercial SoC-Cluster server with five popular DNNs and five datasets that are representative of edge cloud scenarios. We also compare SoCFlow to six competitive baselines [20, 50, 64, 73, 76, 85, 87], including traditional Parameter Server and Ring-AllReduce topologies, as well as advanced methods with hierarchical architecture and gradient compression. The experiments show that SoCFlow can significantly and consistently outperform all baselines in terms of training speed while preserving the convergence accuracy (<1% loss), e.g., 1.6×–740× convergence speedup with 32 SoCs. In most cases, SoCFlow is the only approach that can complete the training within a typical idle time frame of a day (~4hrs), allowing for model updates on a daily basis. Besides, compared to a commodity GPU (i.e., NVIDIA V100) that is widely used for DL training, SoCFlow achieves similar training speed but with 2.31×–10.23× reduced energy consumption.

The major contributions of this work are as follows:

- We highlight the opportunity of co-locating DNN training with existing workloads on deployed SoC-Cluster servers and identify the major challenges through experiments.
- We propose SoCFlow, an efficient DNN training engine for SoC-Clusters. To scale the training speed with more participant SoCs, it incorporates two novel techniques: group-wise parallelism with delayed aggregation and data-parallel mixed-precision training, which enable SoCFlow to fully unleash the heterogeneous SoC hardware capacity under scarce network bandwidth.
- We prototype SoCFlow and evaluate it with extensive experiments. The results demonstrate its superior performance over existing methods.

2 Background and Motivation

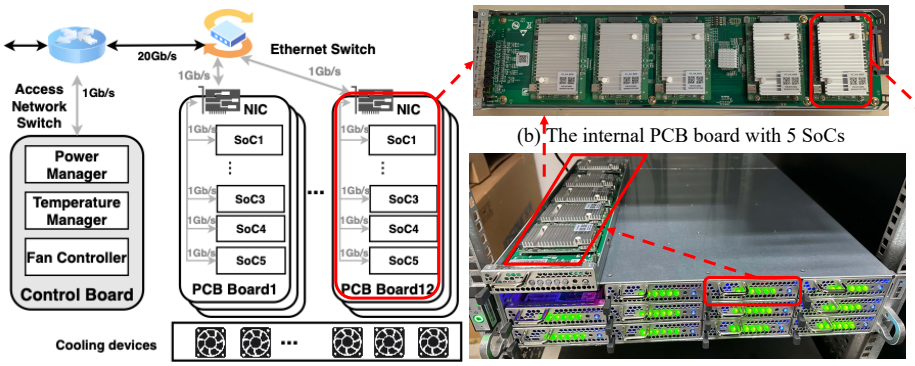
2.1 Edge SoC-Cluster

Physically, a typical computer server in a data center or edge server consists of many-core CPUs and domain-specific accelerators like NVIDIA GPUs and Google TPUs [3, 102]. The processors inside are powerful and monolithic, i.e., hundreds of cores in a CPU chip or even thousands of CUDA cores in NVIDIA GPU. Recently, another unique form of server has emerged in edge data centers, namely SoC-Cluster [14]. One SoC-Cluster consists of massive, low-power ARM-based system-on-chips (SoCs), which are conventionally used only in embedded devices like smartphones and IoTs.

Figure 2 depicts the conceptual architecture, physical structure, and detailed SoC specifications of a commercialized SoC-Cluster we have purchased. The cluster comprises 60 Qualcomm Snapdragon 865 chips [2] and a control board. Each chip in the cluster has an octa-core Kryo 585 CPU, an Adreno 650 GPU, a Hexagon 695 digital signal processor (DSP) that can function as a type of NPU, and 12GB LPDDR5 DRAM. These chips are mounted on 12 printed circuit boards (PCBs), with five on each, and have a 1 Gb/s bandwidth between PCBs and the Ethernet switch and between each SoC and PCB NIC using SAS [1]. All SoCs in a PCB share the same NIC for accessing SoCs from other boards or user clients. The 12 PCBs are also connected to the control board and the access network through a 20 Gb/s bandwidth switch using dual SPF+ interfaces. The control board manages the power, temperature, and fan frequency.

The reasons for the emergence of SoC-Cluster are multifold.

- Firstly and perhaps most importantly, the use of SoCs with the same hardware architecture as smartphones allows for the support of unmodified mobile operating systems (e.g., Android) and mobile applications (e.g., games). This facilitates developers in deploying their cloud services without modifying their codebase for different hardware platforms, especially for graphic-intensive operations. One of the most beneficial applications is mobile cloud gaming [4–7, 13], which hosts gaming logic and rendering on remote servers and streams the rendering results to thin clients. Indeed, mobile cloud gaming has been the dominant workload hosted on SoC-Cluster in the wild [102].
- Secondly, SoC-Cluster delivers denser hardware resources. For instance, the Snapdragon 865-based SoC-Cluster mentioned above has 480 CPU cores in a 2U rack, which is generally higher than conventional servers (e.g., 64 CPU cores for an Intel Xeon server). This indicates that SoC-Cluster can support the same amount of tasks with much less space, which is crucial for space-constrained edge environments like base stations [102, 108].
- Thirdly but not least, SoCs are designed to be energy-efficient by using smaller transistors (≤7nm) and a simple instruction set. Meanwhile, energy efficiency is another critical criterion of edge server design [55, 102], as they



(a) The architecture of the edge SoC-cluster

(b) The internal PCB board with 5 SoCs

(c) The overview of a physical server

CPU	1 x 2.84 GHz Kryo 585 + 3 x 2.42 GHz Kryo 585 + 4 x 1.8 GHz Kryo 585
GPU	Qualcomm Adreno 650
NPU	Hexagon 698 DSP
Memory	12GB LPDDR5
Disk	UFS 3.1 256GB
OS	Android 10 (Kernel 4.19.81)
Network Interface	1 x 1GE RJ45 Port (1Gbps) + 2 x 10GE SPF+ Port (2 x 10Gbps)
Physical Interface	2 Rack Unit

(d) The hardware specification of one SoC

Figure 2. The overview of a typical edge SoC-Cluster that has been widely deployed in a real environment.

are deployed in proximity to populations where electricity is more limited and expensive than large data centers.

2.2 DNN Training on SoC-Clusters

According to our industrial partner (a major edge service provider), tens of thousands of such SoC-Clusters have been deployed on their edge clouds, serving edge applications like mobile cloud gaming. On a daily basis, millions of game sessions are being launched on those servers. The number of those SoC-Clusters is still increasing rapidly because they have demonstrated superior performance to support tasks offloaded from mobile devices.

However, the average CPU utilization of those deployed SoC-Clusters is still low, i.e., below 20% according to the runtime traces we collected. The primary reason is that the workloads hosted on SoC-Clusters exhibit significant tidal phenomena. For example, the number of active game users from 11:00 AM to 17:00 PM is more than one order of magnitude higher than 3:00 AM to 8:00 AM, as shown in Figure 3. This phenomenon attributes to the inherent, user-centric characteristics of the workloads hosted on SoC-Clusters. This observation is also consistent with the recent large-scale, empirical study on commercial edge platforms [102, 104].

To increase the hardware utilization, a typical method is to co-locate best-effort workloads with those latency-critical workloads on servers [71, 100, 112]. In this work, we seek to perform DNN training [112] on the SoC-Clusters when they are idle, regarding its popularity and predictability as discussed in §1. It can also reduce the cost of purchasing additional hardware (e.g., NVIDIA GPUs) to handle the training workloads for edge service providers, thus eliminating the CO_2 emission while manufacturing such saved electronic devices. At runtime, it further reduces the energy consumption for DNN training tasks compared to commodity datacenter-scale GPUs, due to the high energy efficiency of mobile SoCs, as we will experimentally show in §4.

2.3 Challenges and Observations

DNN training is known to be time-consuming; making it shorter, therefore has become a hot topic of cloud computing research in recent years [19, 20, 46, 59, 76, 85]. The significance of fast training is especially valued on SoC-Clusters: an excessively prolonged training task, lasting for tens of hours, not only delays the model's availability for use by users but also adds complexity to software design. This is because the extended training process may occupy multiple idle time windows of SoCs, making it more challenging to manage and optimize system resources effectively.

As the first attempt, we tested classical DNN models (e.g., VGG-11 [86] and ResNet-18 [42] on CIFAR-10 dataset [58]) atop the SoC-Cluster presented above, using the state-of-the-art mobile training engines `MNN` [53]. We use those small to medium-sized models since the models trained on edge servers are often to be deployed on end devices. In our experiments, we aimed to answer the following questions: (1) Is a single SoC adequate to train DNNs fast? (2) If not, can multiple SoCs be used together to speed up training, and how scalable is this approach? (3) How much can heterogeneous processors equipped on SoCs help, especially the mobile NPUs [101]? The results are illustrated in Figure 4.

- **Observation #1: DNN training on a single SoC is extremely slow.** Our experiments show that it takes more than 29 and 233 hours to train VGG-11 and ResNet-18 on the mobile CPU, respectively, as shown in Figure 4(a). Even with state-of-the-art mixed-precision training algorithms [94], training on a mobile NPU still takes nearly 10 and 36 hours for VGG-11 and ResNet-18, respectively. Such long-time training has to span multiple idle time windows and motivates distributed training on multiple SoCs.
- **Observation #2: The efficiency of distributed training is severely bottlenecked by the cross-SoC network.** Ring-AllReduce and parameter-server communication latency with increasing the number of SoCs is shown in Figure 4(b). In our experiments, a PCB board contains 5 SoCs.

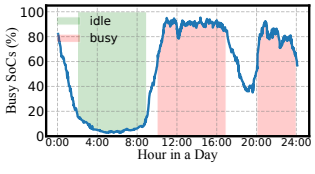
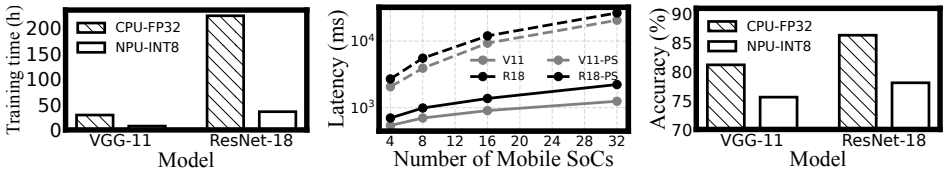


Figure 3. Busy SoCs ratio within a day on deployed SoC-Cluster servers.



(a) End-to-end training time (b) Communication latency (c) Convergence accuracy

Figure 4. Measurement results of training VGG-11 (V11) and ResNet-18 (R18) models on CIFAR-10 dataset atop edge SoC cluster.

Therefore, experiments with less than 5 SoCs involve intra-PCB board communication; otherwise, inter-PCB board communication. Intra-PCB board Ring-AllReduce gradient communication takes 540 and 699 ms to finish for the VGG-11 and ResNet-18 models; parameter-server gradient communication takes 2060 and 2700 ms correspondingly. That is because they are designed for cloud gaming, whose communication is nearly all out-server. Worse, 32-SoC inter-PCB board gradient communication takes 1248, 2225, 20593, and 26505 ms, 2.31–9.81× more than the intra-one. That is because Ring-AllReduce's latency scales linearly with the number of nodes [10, 11, 35, 105], and 32-SoC weight aggregation's preparing and starting the communication for the ResNet18 model takes 1300 ms, 58% of total communication latency. Such delays are unbearable for distributed machine-learning scenarios.

• **Observation #3: mixed-precision training algorithm may degrade model accuracy.** Since most mobile NPUs only support INT8 operations, offloading training tasks to them could lead to accuracy degradation – the price paid for a few times accelerations compared to CPU. Worse, when these INT8 training algorithms are applied to distributed deep learning, the accuracy degradation increases severely. Specifically, the convergence accuracy of training with INT8 on 32 mobile SoC' NPU for VGG-11 and ResNet-18 model is 5.94% and 8.25% lower than training with FP32, as shown in Figure 4(c).

3 SoCFlow Design

SoCFlow is a data-parallel distributed DNN training framework on SoC-Clusters that aims to achieve fast DNN training under the collaboration of many SoCs without compromising accuracy. Figure 5(a) illustrates its overall architecture. SoCFlow takes the training datasets and the DNN to be trained as input. It then continuously trains the DNN until convergence or manually terminated. From the developers' perspective, using SoCFlow is just as easy and standard as using other distributed training frameworks such as TensorFlow or PyTorch. Besides, SoCFlow considers sudden user requests during off-peak periods, e.g., early midnight, when most SoCs are used to train DNN models. SoCFlow includes checkpoints on Mobile SoCs to ensure that a new user-related workload request can preempt training tasks and maintain high service quality for users. Since the group-wise training structure is flexible (§ 3.1), SoCFlow only needs to terminate a logical group of SoCs to minimize the reduction in training efficiency while preserving the convergence accuracy.

SoCFlow mainly consists of two modules:

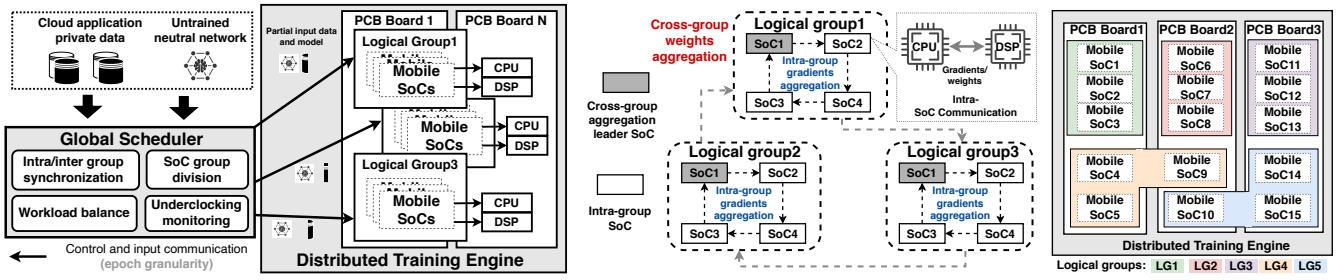
• **Global scheduler** is a lightweight software deployed on the SoC-Cluster's control board. Its primary function is to coordinate the training process. Ahead of the training, it determines how SoCs will be orchestrated, such as SoC grouping, gradients synchronizing frequency, and aggregation methodology, as will be elaborated in §3. Then, it dispatches the training data and model to each SoC. During training, each SoC loads only a partial dataset based on the data-parallelism strategy. This module contains most of SoCFlow' designs.

• **Distributed training engine** is responsible for the gradient computing on each SoC. It supports FP32-based training on CPU, Int8-based training on mobile NPU, or mixed-precision training with both. It also aggregates the gradients/weights sent from other SoCs and synchronizes them.

3.1 Group-wise Parallelism with Delayed Aggregation

In general, there are two ways to address the network bottleneck in distributed DNN training.

- One is to design an efficient network topology, which specifies how data (model weight updates in our case) flows and aggregates across SoCs. In data centers, Ring-AllReduce [85] is a bandwidth-optimal communication strategy and is widely used in network-constrained scenarios. However, it is still inadequate for SoC-Cluster, as previously shown in Figure 4. In essence, it attributes to the severe mismatch of compute-to-network capacity on SoC-Cluster.
- The other is to delay the weights aggregation from each compute node. This approach is commonly used in federated learning [26, 40, 57, 73], where the clients are geo-distributed and are connected to a central server through a wireless network. For instance, FedAvg [73] protocol lets each client train a model for one or many data epochs instead of one batch before uploading it to the cloud for aggregation. By increasing the computing time, the network

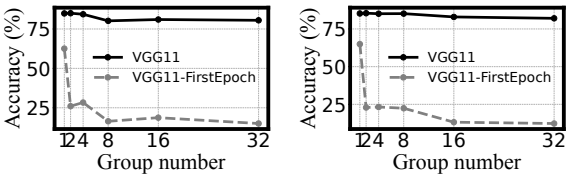


(a) A simplified workflow. (b) Communication topology. (c) Logical-to-physical mapping.

Figure 5. The overview of SoCFlow.

Terms/Symbols	Description
Physical group (PG)	The SoCs that reside in the same PCB.
Logical group (LG)	The SoCs that exchange weights frequently through Ring-AllReduce topology. It is determined by SoCFlow at runtime.
Communication group (CG)	A few logical groups whose intra-group synchronization does not incur NIC contention.
M	The total number of the SoCs.
N	The number of the logical groups.
K	The number of the PCB boards (physical groups).
NUM_{sample}	The number of dataset samples.
BS_g	The sum of all local mini-batch sizes of SoCs within a logical group.
$AC_N^{BS_g}$	N logical groups' convergent accuracy with a global batch size BS_g.

Table 1. The symbols and terms used in §3



(a) VGG11 (b) ResNet18

Figure 6. The testing accuracy after achieving final convergence and for only the first epoch is compared across different group sizes.

bottleneck is mitigated. This approach, however, causes model staleness and potential accuracy degradation [73].

The network capacity of SoC-Cluster lies somewhere between the high-speed data center and wireless network. Therefore, we propose a hierarchical topology that enables group-wise parallelism across SoCs, as shown in Figure 5(b). SoCFlow divides the SoCs into *logical groups*: (1) Within a logical group, the SoCs form a Ring-AllReduce topology and exchange their model updates frequently, i.e., per batch. Different groups' intra-group training can execute in parallel. To ensure similar training accuracy as standard local stochastic gradient descent (SGD) [67, 82], SoCFlow employs synchronized stochastic gradient descent (SSGD) algorithm within each group. SoCFlow leverages both SoC CPU and NPU for model training. Therefore, before cross-SoC synchronization, the gradients computed by the CPU and NPU are aggregated first on the chip. (2) Across logical groups, the weights are aggregated in a delayed manner and infrequently, i.e., per epoch. Unlike federated learning, SoCFlow can shuffle the input data among different groups to guarantee high convergence accuracy. Notably, at the beginning of inter-group synchronization, to reduce synchronization time, each logical group selects a leader (SoC in brown in Figure 5(b)) to aggregate weights with other groups, and all groups' leaders also form a Ring-AllReduce topology.

There are three crucial steps to efficiently realize the proposed mechanism: (1) determining the number of SoCs of each logical group, i.e., the group size; (2) mapping the logical topology of SoC groups into the physical SoC-Cluster architecture; (3) planning the group-wise communication to minimize the NIC contention during training. The terms used in this section are shown in Table 1.

Determining group size. Supposing M SoCs will be divided into N groups, and each group's global batch size is BS_g, the per-epoch training time can be formulated as

$$T_{epoch} = \frac{NUM_{sample}}{(N * BS_g)} * (T_{train}^{BS_g} * \frac{N}{M} + T_{sync}) \quad (1)$$

where $T_{train}^{BS_g}$ and T_{sync} are the computing and synchronization time. In the SoCFlow, T_{sync} consists of intra-group communication and inter-group communication time. Since the computing, intra-group communication, and inter-group communication time are constant, T_{epoch} is negatively correlated to N. Meanwhile, convergence accuracy exhibits a negative correlation with the number of groups, as an increased batch size tends to adversely affect convergence accuracy [38, 44, 106]. This is also confirmed by our experiments in Figure 6, which shows a too large group number notably degrades convergence accuracy.

To identify the largest group size N that guarantees minimal training time while preserving convergence accuracy, SoCFlow offers system designers the flexibility to empirically select this parameter by default and provides an optional heuristic approach. This approach capitalizes on an observation: the training accuracy observed during the initial epoch

closely mirrors the behavior of convergence accuracy, as shown in Figure 6. Thus, during the warm-up stage, SoCFlow profiles the training accuracy from a smaller group size to a larger one. It halts at the first group size where accuracy falls significantly, typically to around 15%, signifying substantial degradation. This is exemplified by the choices of 4 and 8 in Figure 6(a) and (b).

Our experiments validate the efficacy of this heuristic approach. Nevertheless, it is essential to note that this approach relies on heuristics rather than a solid theoretical foundation. Consequently, its applicability across all model types may be limited.

Mapping the logical topology into the physical SoC-Cluster architecture. As shown in §2.1, the SoCs in a SoC-Cluster server are organized into different physical groups (PCBs). Intuitively, a logical group is better mapped to SoCs within the same physical PCB, so that the intensive intra-group data exchange does not go through the external NIC to minimize the contention. Yet, the logical and physical group sizes are often unequal, so that a mixed partitioning is unavoidable.

We first formulate the mapping problem: suppose there are K PCB boards, and each logical groups contains $\frac{M}{N}$ SoCs. The i-th PCB board contains S_i logical groups, denoted as $L_i = \{L_{i,0}, L_{i,1}, \cdots, L_{i,S_i}\}$. If the number of SoCs for the j-th logical group in i-th PCB board is smaller than $\frac{M}{N}$ (denoted by $|L_{i,j}| < \frac{M}{N}$), there must be at least one SoC in other PCB boards, therefore incurring inter-PCB communication. We use L_i^{inter} to represent logical groups with inter-PCB communications inside i-th PCB in Eq 2.

$$L_i^{inter} = \{x \mid |x| < \frac{M}{N}, \forall x \in L_i\} \tag{2}$$

The maximum number of K PCB boards' inter-PCB communication logical groups is denoted by C,

$$C = max\{|L_i^{inter}|, \forall i \in K\} \tag{3}$$

which represents the conflict numbers. SoCFlow's objective is to minimize C.

To solve the problem, SoCFlow employs a novel mapping algorithm: *integrity-greedy mapping*: First, SoCFlow maps as many logical groups as possible to physical groups without splitting. Figure 5(c) illustrates an example with a logical group size 3 and a physical group size 5. In this case, each three nodes within logical groups 1–3 are all placed within the first three SoCs in the corresponding PCBs. Second, the rest of the logical nodes are mapped in sequence. For both logical and physical nodes, we squeeze them into a 1-D dimension by placing the nodes within a group continuously, and the mapping follows the squeezed order. In this step, the four nodes within logical group 4 span the 1st and 2nd PCBs, while the nodes within logical group 5 span the 2nd and 3rd PCBs.

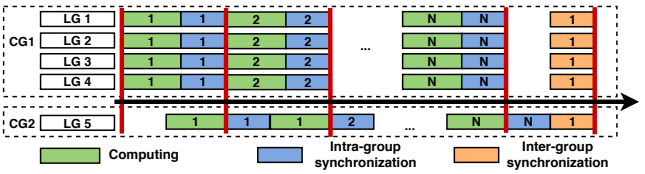


Figure 7. The group-wise communication planning used by SoCFlow. *CG* and *LG* represent the communication group and logical group, respectively.

We have the following theorems for the *integrity-greedy mapping* algorithm. The first theorem can be proven by the "greedy stays ahead" algorithm [8]. Mapping as many LGs as possible with no inter-PCB communication always has less NIC contention. While the second one can be proved by contradiction.

Theorem 1: *Integrity-greedy mapping minimizes C.* This theorem guarantees the optimality for the mapping stage.

Theorem 2: *Integrity-greedy mapping guarantees that each logical group contends with up to two other logical groups for NIC.* This theorem is used in the next stage in planning communications.

Planning group-wise communication to minimize the contention. When the logical group size does not match with the physical group size (which is often true), the NIC contention between logical groups is inevitable. To mitigate such contention, SoCFlow seeks to carefully determine their communication timing.

Specifically, SoCFlow further combines logical groups into different *communication groups (CGs)*. In the same CG, different logical groups' intra-group synchronization is interleaved. For example, logical groups 1–4 in Figure 5(c) forms one CG, while logical group 5 is put into another CG. This is because LG1–3 have no inter-PCB communication and can be placed anywhere, while LG4 and LG5 have inter-PCB communication and are conflicted with each other. After CG division, different CGs' intra-group synchronization communicates separately in sequence to avoid network contention, as shown in Figure 7. Specifically, designing an effective communication strategy faces the following two challenges:

- *How to divide logical groups into CGs.* In general, finding the minimum CGs division is crucial to SoCFlow, since more CGs need more communication intervals. However, such a problem is equivalent to minimum graph coloring problem [79], which is an NP-Hard problem.
- *How to plan CGs communication sequence to minimize the idle time of processors.* Since only one CG's logical groups can synchronize at some time, other CGs should wait until no network communication. The waiting time may lead to wasting processor resources.

To address the first challenge, fortunately, theorem 2 of *integrity-greedy mapping* guarantees that one logical group contends for NIC with up to two other logical groups. This

transforms the CG division problem into the minimum bipartite graph coloring problem, for which the optimal solution can be obtained using the depth-first search (DFS) [22].

Regarding the second challenge, an intuitive approach is to use a pipeline of several CGs to hide the intra-group communication time. In general, to completely hide n sequences of communication, the computing time should be at least $n-1$ times longer than the communication time. Fortunately, the solution to the first challenge [22] guarantees that the number of CGs needed is at most two. Therefore, the communication can be totally hidden as long as the computing is slower than the communication, which is observed to be true in most of our experiments. Figure 7 illustrates how the overlapping works.

Lastly, after training all batch samples, all SoCs start inter-group synchronization. Therefore, the extra delay of SoCFlow is only one intra-group and inter-group synchronization time.

3.2 Data-parallel Mixed-precision Training

As discussed in §2.3, mobile NPU can accelerate DNN training at the cost of compromised accuracy by using INT8 data format. To address this, SoCFlow exploits both the CPU and NPU on mobile SoCs in parallel, i.e., a mixed-precision training paradigm, to achieve both high accuracy and fast training. Per batch, the training data is partitioned into CPU and NPU. When training completes on both CPU and NPU, SoCFlow directly aggregates the weights from them through on-chip IPCs before the intra-group synchronization, as shown in Figure 5(b). Currently, we employ the standard SGD as the training optimizer on CPU and the state-of-the-art INT8-based optimizer [94] on NPU.

SoCFlow tackles two primary issues in designing the mixed-precision training algorithm: (1) The numerical errors incurred by FP32-to-INT8 quantization on NPU could accumulate exponentially as training goes on. Therefore, naively averaging the gradients from the CPU and NPU could lead to significant accuracy loss. SoCFlow needs to minimize the quantization errors to guarantee convergence accuracy. (2) Unlike distributed training scenarios where worker nodes are homogeneous [59, 63, 64, 80], mobile CPUs/NPUs face huge training speed gaps. SoCFlow needs a way to harmoniously pace the two training processes.

To solve the above two problems, SoCFlow controls the relative amount of data fed to the models running on CPU and NPU, without re-engineering the network structure or training process. More specifically, SoCFlow introduces two metrics:

- α – *confidence* that indicates the error gap between the INT8 model and the FP32 model. To calculate it, SoCFlow simply profiles the validation set on CPU/NPU prior to each training epoch. It can be formulated as

$$\alpha = Cos(< logits_{FP32}, logits_{INT8} >) \tag{4}$$

We use cosine similarity since it avoids the negative effect of the varied gradients' magnitudes from FP32 and INT8, as shown in Eq 4. When α approaches zero, the INT8 model is less accurate, so SoCFlow allocates more data for CPU training to mitigate training accuracy loss; otherwise, more data should be fed to the NPU to improve training speed. Typically, the cosine similarity of two models' logits decays exponentially [123]. It means as the error gap between the two models increases, the decline of α becomes slower. Thus, although the minor quantization error will accumulate exponentially after massive multiplication and addition calculation, α does not decrease much. Correspondingly, SoCFlow leverages $e^{-\alpha}$ to control input data partition to avoid exponential decay. The portion of mini-batch samples into the CPU model should be no less than $e^{-\alpha}$, while the portion into the NPU model must not exceed $1 - e^{-\alpha}$. Weight aggregation is also modified as follows.

$$w_{t+1} = e^{-\alpha} * w_{t+1}^{FP32} + (1 - e^{-\alpha}) * w_{t+1}^{INT8} \tag{5}$$

where w_{t+1}^{FP32} and w_{t+1}^{INT8} are $t+1$ iteration weights from the FP32 and INT8 model, respectively. Oftentimes, at the beginning of a training task, the INT8 model on NPU is accurate enough and α is close to 1, so much of the training data is fed into the NPU to improve the training speed; when approaching convergence, α is close to 0 so more data is fed to the CPU to guarantee the convergence accuracy.

- β - *compute power ratio* that represents the ratio of compute power for heterogeneous processors. It is simply profiled as the CPU-to-NPU performance gap before the training task begins. For SoCFlow, β can be formulated as

$$\beta = \frac{T_{NPU}}{T_{NPU} + T_{CPU}} \tag{6}$$

To avoid processor idleness, the portions of input data being fed into the NPU should be exactly as β does.

When jointly considering accuracy requirements and performance issues, SoCFlow always feds $max\{e^{-\alpha}, 1-\beta\}$ portion of data into the CPU. That is because when $e^{-\alpha}$ is larger than β, NPU processing capacity is not enough and the CPU is idle, so feeding more data into NPU cannot gain any benefit. Otherwise, SoCFlow is bottlenecked by the quantization error from the INT8 model, and more data should be fed into the CPU even if the NPU is idle.

4 Evaluation

4.1 Implementation and Setups

We have fully implemented SoCFlow with 5.2k LoC in C/C++. The prototype is a standalone framework supporting models exported from TensorFlow [18] and Pytorch [80]. SoCFlow leverages MNN [53] (the most lightweight on-device training framework) as the CPU backend and Mandheling [101]

Model	Dataset	Learning methods
LeNet [62]	EMNIST [27]	From scratch
	Fashion-MNIST [99]	
VGG-11 [86]	CIFAR-10 [58]	
	CelebA [72]	
ResNet-18 [42]	CIFAR-10	
	CelebA	
MobileNet_V1 [45]	CIFAR-10	
ResNet-50 [42]	CINIC-10 [32]	Transfer learning

Table 2. DNN models used in the experiments.

as the NPU backend. We follow PyTorch to implement gradient synchronization, such as layer-by-layer computing-communication overlapping, and aggregation in the backward pass and optimizer. All the network communication, including Ring-AllReduce, parameter server, and federated learning, are implemented over TCP protocol. Parameters and weights aggregation through CPU and NPU are over shared memory for efficiency. In addition, we have implemented two key optimizations to make `SoCFlow` more efficient and practical: (1) Gradient computing-communication overlap to mitigate synchronization delays; (2) Underclocking-aware workload re-balancing to address potential performance degradation.

Hardware setup. We test the performance of `SoCFlow` on the SoC-Cluster as discussed in §2.1. All devices run Android OS 10. By default, we always run the baselines on 4 BIG CPU cores. The CPU frequency is controlled by the OS's dynamic voltage and frequency scaling (DVFS) controller. The physical, logical, and communication groups used in the experiments are 5, 8, and 2, respectively.

Models and datasets. We test with a range of typical CNN models with various datasets: LeNet [58], VGG-11 [86], ResNet-18/50 [42], and MobileNet_V1 [45], as listed in Table 2. The input data for LeNets are either EMNIST or Fashion-MNIST (input size 28*28), while for VGG-11, ResNet18 and MobileNet_V1 are either CIFAR-10 or Celeba (input size 32*32). In addition to training from scratch, `SoCFlow` also evaluates the transfer learning scenarios: finetuning on CIFAR-10 while pre-trained on CINIC-10 dataset (same categories with 40k more images) with ResNet-50. We choose small to medium-sized models since the models trained on edge servers are often to be deployed on end devices.

Baselines. We compare `SoCFlow` with 6 baselines which can be divided into two categories:

• *Distributed machine learning baselines* (1) Parameter Server (`PS`): the traditional FP32-based centralized aggregation method [64]. (2) Ring-AllReduce (`RING`): the traditional FP32-based allreduce training method following the workflow of Horovod [85]. (3) HiPress [20]: a compression-aware gradient synchronization framework for data-parallel DNN training. It uses DGC [69] as the sparsification compression algorithm. (4) 2D parallelism [87] (`2D-Paral`): a hierarchical topology with node grouping. Across groups, it exploits Ring-AllReduce-based data parallelism; within a group, it exploits pipeline parallelism as PipeDream [76]. We do not compare 3D parallelism [87] as its tensor parallelism is more suitable for large models like GPT3 [78], while `SoCFlow` is designed for small to medium-sized models on edges.

• *Federated learning baselines* (1) Federated learning (`FedAvg`): the traditional FP32-based federated learning protocol [73]. (2) Tree-aggregation-based hierarchical federated learning (`T-FedAvg`): It divides SoC clients into several groups and exploits tree-based hierarchical aggregation federated learning protocol [50, 74] with FedAvg algorithm. Both of the baselines have been implemented within the context of independent and identically distributed (IID) settings.

To make the comparison fair, all baselines are enhanced with the two optimizations in §4.1 if applicable.

Metrics. We mainly measure convergence accuracy, training time, and energy consumption during training. The energy consumption is calculated through SoC-Cluster's control board power management system. All experiments are repeated three times and we report the average numbers.

4.2 End-to-end Performance

Overall performance. We comprehensively investigate the end-to-end training performance of `SoCFlow` using 32 SoCs. The convergence accuracy, training time, and energy consumption of the 32 SoCs are illustrated in Table 3, Figure 8, and Figure 9. Except for MobileNet_V1 uses a global batch size of 256, other models all use 64. Our key observation is that **`SoCFlow` consistently and remarkably outperforms other baselines on training time and energy consumption with negligible accuracy loss.**

• *Training time of* `SoCFlow` *v.s. distributed machine learning baselines.* Compared with industrial baselines, like `PS` and `RING`, `SoCFlow` achieves a 94.4–740.7× and 14.8–143.7× speedup, respectively, as shown in Figure 8, with negligible accuracy degradation, e.g., <1%, in most cases, as shown in Table 3. Those benefits from our group-wise parallelism with delayed aggregation and mixed-precision data-parallel training algorithm. The first technique can reduce communication overhead, while the second one can exploit NPU fully without influencing accuracy.

Besides, compared with state-of-the-art baselines, `HiPress`, and `2D-Paral`, `SoCFlow` can still reduce training time by 7.4–98.2× and 4.4–50.4×, respectively. That is because although such baselines can decrease communication volume or increase parallelism, they still cannot avoid inter-PCB communication contention. While `SoCFlow`'s group-wise parallelism with delayed aggregation can exploit logical-to-physical topology mapping and communication planning to mitigate with no network contention when synchronizing weights.

Last, `SoCFlow` guarantees that all training tasks finish within two hours smaller than the SoC-Cluster's idle time (below the dark red line in Figure 8) so that the model can be

Model	Local	PS		RING		2D-Paral		HiPress		FedAvg		Tree-FedAvg		Ours	
	Acc.	Acc.	Degrad.	Acc.	Degrad.	Acc.	Degrad.	Acc.	Degrad.	Acc.	Degrad.	Acc.	Degrad.	Acc.	Degrad.
MobileNet	88.5	87.9	**-0.6**	87.9	**-0.6**	87.9	**-0.6**	87.9	**-0.6**	85.4	**-3.1**	85.4	**-3.1**	88.7	**0.2**
VGG11	84.5	84.4	**-0.1**	84.4	**-0.1**	84.4	**-0.1**	84.4	**-0.1**	80.4	**-4.1**	80.4	**-4.1**	82.2	**-2.3**
ResNet18	87.7	87.3	**-0.4**	87.3	**-0.4**	87.3	**-0.4**	87.3	**-0.4**	82.1	**-5.6**	82.1	**-5.6**	84.5	**-3.2**
VGG11-CelebA	96.9	96.9	**0**	96.9	**0**	96.9	**0**	96.9	**0**	96.8	**-0.1**	96.8	**-0.1**	97.1	**0.2**
Resnet18-CelebA	97.3	97.4	**0.1**	97.4	**0.1**	97.4	**0.1**	97.4	**0.1**	97.4	**0.1**	97.4	**0.1**	97.2	**-0.1**
LeNet5-EMNIST	87.5	87.6	**0.1**	87.6	**0.1**	87.6	**0.1**	87.6	**0.1**	85.6	**-1.9**	85.6	**-1.9**	87.7	**0.2**
Lenet-FMNIST	91.6	91.6	**0**	91.6	**0**	91.6	**0**	91.6	**0**	90.7	**-0.9**	90.7	**-0.9**	91.1	**-0.5**
ResNet50-Finetune	69.9	69.9	**0**	69.9	**0**	69.9	**0**	69.9	**0**	x	**x**	x	**x**	68.9	**-1**
Average degradation			**-0.16**		**-0.16**		**-0.16**		**-0.16**		**-2.23**		**-2.23**		**-0.81**

Table 3. A summary of end-to-end training convergence accuracy. "Acc.": accuracy; "Degrad.": accuracy degradation.

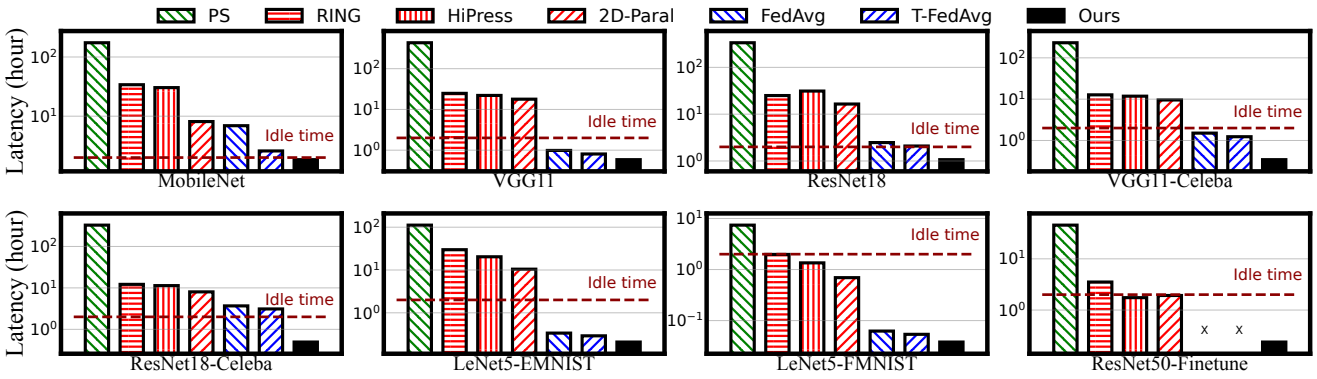


Figure 8. End-to-end training time up to convergence under different training scenarios.

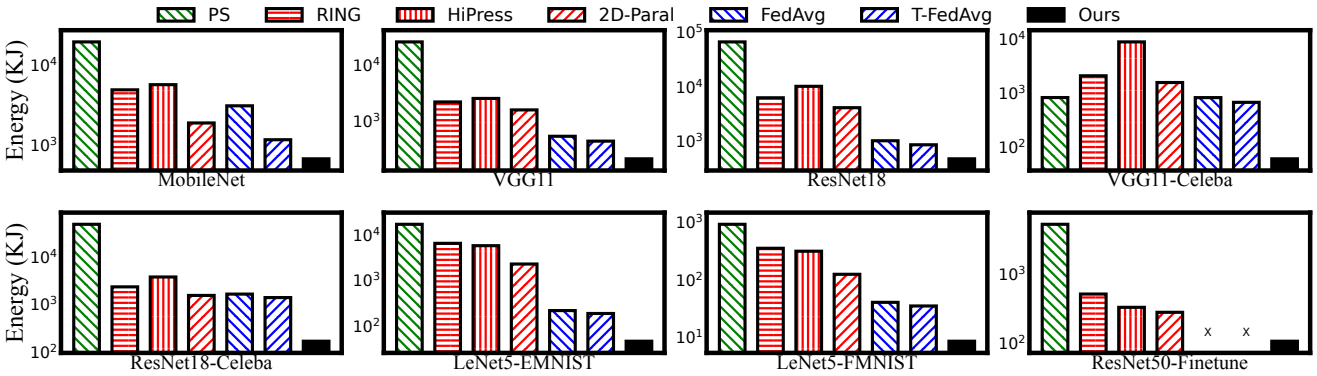


Figure 9. End-to-end training energy consumption up to convergence under different training scenarios.

updated and applied to cloud applications every day; while no distributed machine learning baselines can satisfy such strict deadline. Therefore, SoCFlow can both boost training efficiency and be practical in use.

• *Energy consumption of* SoCFlow *v.s. distributed machine learning baselines.* As shown in Figure 9, SoCFlow's improvements in energy consumption are also impressive as in training speed. SoCFlow can reduce energy consumption by 20.0–158×, 1.9–60.2×, 3.1–144.3×, and 2.6–49.8× for PS, RING, HiPress and 2D-Paral, respectively. That is because (1) the existing distributed deep learning algorithms lead to long-time synchronization communication, wasting lots of energy. (2) SoCFlow can leverage energy-efficient NPU to accelerate training speed and reduce energy consumption.

• SoCFlow *v.s. federated learning baselines.* Compared with the federated learning method, SoCFlow achieves an accuracy improvement of 0.1–3.3%. The accuracy enhancement is attributed to SoCFlow mitigating the precision loss associated with INT8-based training by offloading a portion of the training workloads to the CPU in FP32 format. Additionally,

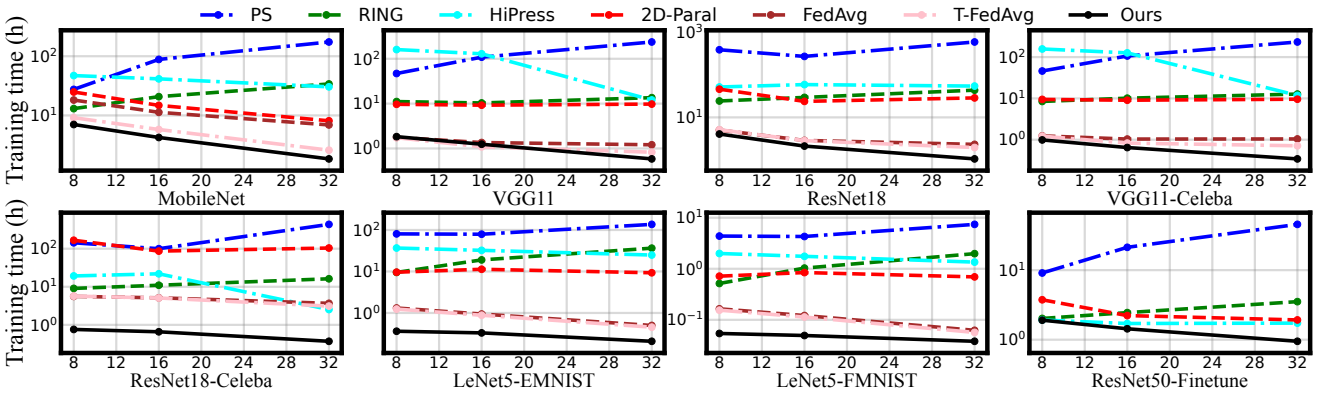


Figure 10. Elapsed training time taken to reach same target accuracy under various SoC numbers.

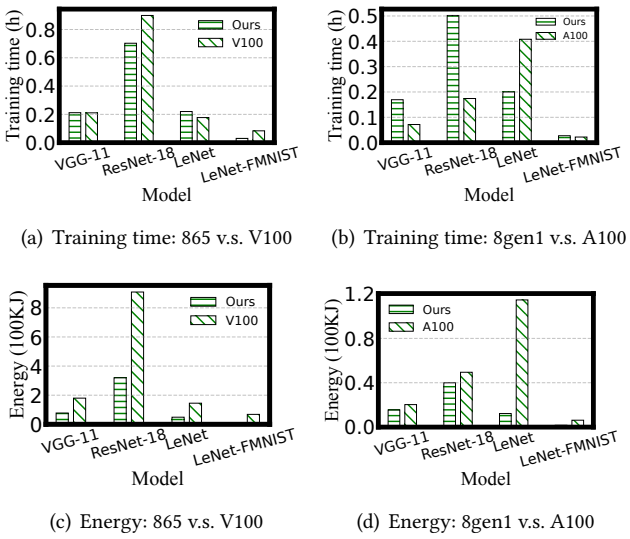
SoCFlow substantially reduces training time, achieving an average speedup of 2.85x for FedAvg and 2.17x for T-FedAvg. This efficiency gain can be primarily attributed two reasons: (1) While federated learning baselines are not constrained by communication bottlenecks, they grapple with gradient staleness and require more epochs to converge to the same accuracy. SoCFlow addresses this issue by implementing a synchronized gradient updating approach. (2) SoCFlow harnesses NPUs to accelerate the training process while maintaining high accuracy.

In addition, SoCFlow reduces energy consumption by 2.1–9.9 and 1.7–11.0×, compared with FedAvg and T-FedAvg, respectively, as shown in Figure 9. The benefits come from the high energy efficiency of NPU and accelerated convergence.

4.3 Scalability

We comprehensively investigate the scalability training performance of SoCFlow under 8, 16, and 32 SoCs. Figure 10 shows the training time trend reaching the same accuracy (99% relative convergence accuracy, e.g., 87% for MobileNet_V1) when involving more SoCs in learning tasks. We do not include the results of ResNet50-Finetune using FL-based baselines, as it did not converge. Except for MobileNet_V1 uses a global batch size of 256, other models all use 64. Our experiments show that **SoCFlow consistently outperforms all baselines from 8 to 32 SoCs, and its benefits are more prominent with the increasing SoC number.**

SoCFlow reduces training time on 8 SoCs by 83.3×, 8.89×, 2.31×, 36.4×, 2.51×, and 53.8× on average for PS, RING, HiPress, 2D-Paral, FedAvg, and T-FedAvg, respectively; while the speedups for 32 SoCs are 474.8×, 49.3×, 2.35×, 52.8×, 3.1× and 35.7× correspondingly, which are 2.6 × larger than that of 8 SoCs on average. That is because our group-wise parallelism with delayed aggregation is flexible and scalable, not increasing the network congestion with the SoC number increasing.



(a) Training time: 865 v.s. V100 (b) Training time: 8gen1 v.s. A100

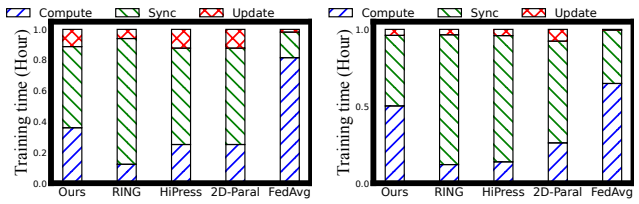
(c) Energy: 865 v.s. V100 (d) Energy: 8gen1 v.s. A100

Figure 11. Training time and energy consumption comparison between SoCFlow and traditional datacenter GPUs using PyTorch.

4.4 Comparison with Traditional Datacenter GPU

In this section, we examine the performance of SoCFlow in comparison to traditional datacenter GPUs. It's important to note that our objective in these experiments is not to present SoC-Cluster as a superior alternative to traditional datacenter GPUs. Rather, our goal is to illustrate how SoCFlow can effectively utilize the available resources in SoC-Clusters, particularly when dealing with small models as the focus of this study.

We conducted a comprehensive assessment on the end-to-end training performance of SoCFlow under 60 SoCs, in contrast to a standard server GPUs, using PyTorch, as shown in Figure 11. We compared Snapdragon 865 SoC with the NVIDIA V100, and the latest Snapdragon 8gen1 SoC with the NVIDIA A100. Our selection is mainly based two main



(a) VGG-11 (b) ResNet-18

Figure 12. A breakdown training time under VGG-11 and ResNet-18 models on CIFAR-10.

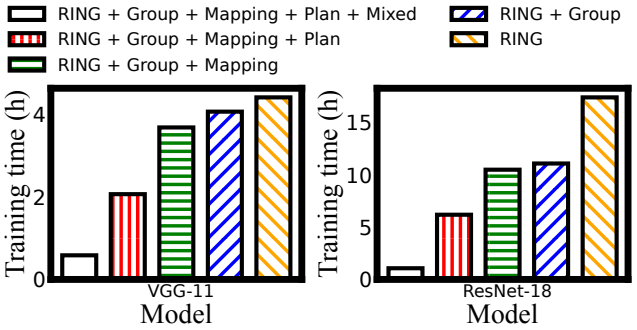


Figure 13. Ablation study of the hierarchical weights aggregation.

reasons: (1) Relatively consistent performance gap in latest versions. Despite the Snapdragon 865 SoC being introduced later than the V100, it is noteworthy that the performance improvements of mobile SoCs outpaces those in server GPUs. For instance, the NPU in the 8gen2 (2022) exhibits an 18× increase in performance compared to the Snapdragon 865 SoC [15], while such performance gain from the H100 (latest NVIDIA GPU) to the V100 is only about 9×. (2) It is important to acknowledge that data center-level GPUs such as the V100 are not primarily designed for training small models that often exhibit low GPU utilization. Nonetheless, they are frequently adopted in edge cloud environments to address a variety of training scenarios, including training small-to-medium-sized models. Our experiments show that **SoCFlow achieves similar training speed but with up to 10.23× reduced energy consumption without comprising the convergence accuracy.**

Compared to the V100 GPU, SoCFlow achieves a speedup of 0.80–2.79× for the VGG11-CIFAR10, ResNet18-CIFAR10, LeNet-EMNIST, and LeNet-FMNIST models. This is due to SoCFlow's ability to tap into the computing power of the 60-SoC heterogeneous processors and break network limits to accelerate training speed. In addition, SoCFlow consumes 2.31×, 2.81×, 2.96×, and 10.23× less energy than the V100, respectively. This is because mobile SoCs are generally more energy-efficient than the V100, especially for mobile NPU, and SoCFlow can fully utilize this advantage. The results of the A100 also demonstrates analogous outcomes.

4.5 Breakdown analysis of training time

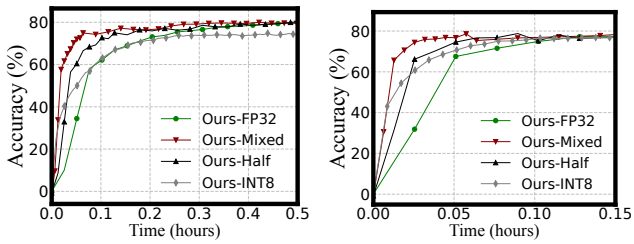
In this section, we conduct a comprehensive investigation into the breakdown of training time consumption. Typically, training time comprises three main components: gradients computing (Compute), gradients/weights synchronization (Sync), and parameter updates (Update). The breakdown results for VGG-11 and ResNet-18 under 32 SoCs are illustrated in Figure 12. **SoCFlow achieves a delicate balance between distributed machine learning baselines and federated learning baselines approaches by trading off weights synchronization time with accuracy.**

The synchronization in RING consistently occupies the most time (e.g., 81% for VGG-11) due to network contention. Two communication-efficient baselines HiPress and 2D-Paral manage to reduce the synchronization overhead to an average of 76.5% and 71.5% on average, respectively. Nevertheless, the bottleneck in these two baselines still persists in communication since they synchronize inter-group (per-batch) gradient communication simultaneously, contending for the physical board NIC. FedAvg's synchronization time is notably lower, only 16.5–34.7%, owing to its epoch-based synchronization strategy. SoCFlow's synchronization time falls in the middle of distributed machine learning baselines and federated learning baseline, accounting for only 46% of the total training time, attributed to its efficient hierarchical weight aggregation.

4.6 Ablation Study

Overall techniques. Figure 13 demonstrates how each technique in SoCFlow is combined together to help train efficiently and scalably step by step. The rightmost bar is the same as baseline RING, while the leftmost one is SoCFlow. The three steps in §3.1 and the data-parallel mixed-precision training algorithm in §3.2 are represented by Group, Mapping, Plan, and Mixed correspondingly.

First, dividing SoCs into groups can avoid the network traffic jam of all SoCs communicating together, leading to an 8% and 57% speedup for VGG-11 and ResNet-18 models, respectively. The more benefits for ResNet-18 are because of more gradients to be synchronized. Besides, mapping the logical topology into the physical SoC-Cluster architecture can reduce training time by 1.05–1.10 ×. That is because our mapping algorithm considers and minimizes the network conflicts in synchronization. In addition, group-wise communication planning can achieve a 1.69–1.78 × speedup since communication planning can avoid NIT conflicts and mitigate the network bottleneck. Last, the data-parallel mixed-precision training algorithm reduces by 3.53–5.78× because



(a) VGG-11 (b) Resnet-18

Figure 14. Ablation study of the mixed-precision data-parallel training algorithm.

it can exploit both the CPU and NPU to train a model in parallel.

Although SoCFlow also has batch synchronization due to its hierarchical weight aggregation, the synchronization time is relatively short, only 25% of that for Ring-AllReduce. On the other hand, FL spends a lot of time on epoch synchronization, while Ring-AllReduce hardly spends any time on it.

Mixed-precision data-parallel training algorithm. Figure 14 further shows how SoCFlow's mixed-precision data-parallel training algorithm can improve convergence speed and accuracy. It only shows the first 10 epochs. Here, we propose three new baselines. *Ours-FP32* only uses CPU to train an FP32 model; while *Ours-INT8* only uses DSP to train an INT8 model. *Ours-Half* always inputs half of the datasets to the FP32 model and the rest to the INT8 model, a special case as SoCFlow ($\alpha = 0.7$), detailed in §3.2.

As shown in Figure 14, **SoCFlow can achieve both high accuracy, similar to *Ours-FP32*, and high training speed, similar to *Ours-INT8***. That is because, with the help of the INT8 model confidence hyper-parameter α and compute power ratio β, SoCFlow can feed the maximum portion of data to the NPU INT8 model to improve training speed without affecting the convergence accuracy. Specifically, α will decrease with training progresses, which means that the INT8 model is less and less confidential. At the beginning of training, more data is trained on the INT8 model to improve training speed with rapid improvement in accuracy as *Ours-INT8* does. At the end of the training, more data to the FP32 model to ensure higher accuracy, similar to *Ours-FP32*. As a result, SoCFlow can incorporate the advantages of *Ours-INT8* and *Ours-FP32* methods. On the contrary, the ad-hoc method, *Ours-Half*, cannot dynamically adjust the portion of input data fed into the CPU and NPU models, so its training speed is slower than *Ours-INT8* and the accuracy is lower than *Ours-FP32*, missing optimization opportunities that SoCFlow can exploit.

5 Discussion

Feasibility of INT8 training on SoC-Cluster. SoC-Cluster is an edge cloud platform primarily tailored to serve relatively small models and simple computer vision tasks, such as face recognition. While INT8 training may not be the prevailing approach, recent extensive efforts on INT8 training [94, 114, 123] have delved into its potential, particularly for these small edge-oriented models. By harnessing the efficiency of mobile SoC NPUs in INT8 operations [15–17], these mixed-precision training algorithms have showcased their ability to significantly reduce on-SoC training time and energy consumption while incurring only negligible accuracy losses [101]. Moreover, SoCFlow introduces a novel *data-parallel mixed-precision training algorithm* designed to mitigate the precision loss inherent in INT8-based training. This algorithm effectively offloads a portion of the training workload to the CPU, utilizing the more precise FP32 format to compensate for the precision loss entailed by INT8-based training. This innovative approach not only maximizes training speed but also preserves training accuracy.

Future applicability of SoCFlow. Recent advancements have highlighted a compelling trend in mobile SoC NPUs, wherein their capabilities have expanded significantly. These NPUs now concurrently accommodate a diverse range of low-precision data formats, including INT4, INT8, INT16, and FP16 [15–17]. These versatile data formats cater to a spectrum of application scenarios, spanning from image classification to keyword detection. For instance, the latest NPUs in Snapdragon 8gen2 support INT8 and FP16 operations, yielding a remarkable 18× speedup over the Snapdragon 865 employed in our experimental setup [15]. Given that SoCFlow is a distributed training framework orthogonal to low-precision training algorithm and leverages two main techniques to train deep learning models fast and scalably without being influenced by the network bottleneck and the accuracy loss of low-precision training algorithm, the recent developments of mobile NPUs opening up more opportunities for SoCFlow to train relatively larger DNNs, including Transformers [90], on SoC-Cluster.

6 Related Work

On-device training. Recently, there has been a trend to train a DNN model on mobile devices locally [9, 26, 40, 57, 73, 95, 101, 103]. DeepType [103] proposes *incremental training* to effectively train a personalized deep learning model from a global model. Melon [95] reduces memory usage by a novel lifetime-aware memory pool and memory-calibrated progressive recomputation. Some of them tried to leverage GPU/DSP offloading to accelerate training speed and reduce training energy consumption [33, 101]. SoCFlow is orthogonal to and compatible with those system-level optimizations.

Mixed-precision DNN training has been proposed to reduce training cost [21, 29, 31, 48, 68, 70, 81, 94, 96, 98, 114, 116,

118, 119, 121–123]. These approaches use lower-precision formats, such as INT8 and INT16, to represent the weights and activation generated during training. UI8 [123] further proposes a cosine-distance-based gradient quantization error estimation technique and direction-aware clip function to minimize gradient quantization error. SoCFlow's data-parallel mixed-precision training algorithm is built on them but algorithm-independent.

Distributed machine learning. Since training DNN models is too time-consuming, nowadays, many distributed training approaches are proposed to speed up the training process, including data parallelism [59, 63, 64, 80], model parallelism [34], hybrid parallelism [52], and pipeline parallelism [19, 46, 76]. Besides, many solutions have been proposed to optimize communication efficiency between different workers [30, 36, 41, 49, 54, 93, 109, 113]. SoCFlow is motivated by those efforts and is the first framework to support distributed training atop edge SoC-Cluster. Furthermore, some researchers propose asynchronous or stale synchronous parallel (SSP) aggregation which allows distributed workers to read older, stale versions of parameters from a local cache, instead of waiting to get them from a central storage [23, 43, 117, 120]. This approach can reduce significantly synchronization waiting time while still providing correctness guarantees.

Federated learning is also an emerging machine-learning paradigm [26, 40, 57, 73, 77, 91] built atop on-device training and requires many clients to train a DNN model collaboratively. Most of the prior works focus on model compression techniques to address the communication bottleneck, while some propose tree aggregation [50] and LAN-WAN aggregation [107] to save network traffic. Those studies inspire SoCFlow group-wise parallelism with delayed aggregation.

Junkyard Computing. A few recent works [75, 88] have explored the opportunity of recycling obsolete smartphones to build a computing cluster. However, they are mostly prototyped with a very limited number of devices and tested with microbenchmarks. In contrast, SoCFlow aims to develop production-ready, widely deployed hardware. Meanwhile, the challenges and opportunities for deep learning training on the SoC-Cluster have not been addressed in those works.

7 Conclusion

This work presented SoCFlow, the first framework that can efficiently train deep learning models on SoC-Cluster. To achieve efficient and scalable training performance under limited cross-SoC network capacity, SoCFlow incorporates two novel techniques, including group-wise parallelism with delayed aggregation and a data-parallel mixed-precision training algorithm. Our experiments have demonstrated that SoCFlow significantly and consistently outperforms all baselines regarding the training speed while preserving the convergence accuracy, e.g., 1.6×–294× convergence speedup with 32 SoCs.

8 Acknowledgement

This work was supported by National Key R&D Program of China (No.2021ZD0113001), NSFC (62325201, 62102045, 62172008), the National Natural Science Fund for the Excellent Young Scientists Fund Program (Overseas), Alibaba Group through Alibaba Innovative Research (AIR) Program, the Beijing Outstanding Young Scientist Program under the grant number BJJWZYJH01201910001004, and Center for Data Space Technology and System, Peking University. Xin Jin is the corresponding author.

A Artifact Appendix

A.1 Abstract

SoC-Cluster, a novel server architecture composed of massive mobile system-on-chips (SoCs), is gaining popularity in industrial edge computing due to its energy efficiency and compatibility with existing mobile applications. However, we observe that the deployed SoC-Cluster servers are not fully utilized, because the hosted workloads are mostly user-triggered and have significant tidal phenomena. To harvest the free cycles, we propose to co-locate deep learning tasks on them.

We present SoCFlow, the first framework that can efficiently train deep learning models on SoC-Cluster. To deal with the intrinsic inadequacy of commercial SoC-Cluster servers, SoCFlow incorporates two novel techniques: (1) the group-wise parallelism with delayed aggregation that can train deep learning models fast and scalably without being influenced by the network bottleneck; (2) the data-parallel mixed-precision training algorithm that can fully unleash the heterogeneous processors' capability of mobile SoCs. We have fully implemented SoCFlow and demonstrated its effectiveness through extensive experiments. The experiments show that SoCFlow significantly and consistently outperforms all baselines regarding the training speed while preserving the convergence accuracy, e.g., 1.6×–740× convergence speedup with 32 SoCs. Compared to commodity GPU (NVIDIA V100) under the same power budget, SoCFlow achieves comparable training speed but reduces energy consumption by 2.31×–10.23× with the same convergence accuracy.

A.2 Artifact check-list (meta-information)

- **Model: LeNet, VGG-11, ResNet-18, ResNet-50, MobileNet-v1**
- **Data set: CIFAR-10, EMNIST, Fashion-MNIST, CelebA, CINIC-10**
- **Run-time environment: Ubuntu 18.04.6**

- **Hardware:** SoC-Cluster
- **Execution: C++ and bash scripts**
- **Metrics:time-to-accuracy**
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: MIT**
- **Archived (provide DOI)?: https://zenodo.org/doi/10.5281/zenodo.8406266**

A.3 Description

A.3.1 How to access. Our code is publicly available at zenodo[1].

A.4 Installation

unzip -d asplos11-AE.zip

A.5 Experiment workflow

cd project/android/MNN-standard-DSP
bash run.sh SoCNumber GroupNumber 101 result.log 1

A.6 Methodology

Submission, reviewing and badging methodology:

- artifact available

References

[1] Serial attached scsi. https://en.wikipedia.org/wiki/Serial_Attached_SCSI, 2019.

[2] Snapdragon 865 5g mobile platform. https://www.qualcomm.com/products/application/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-865-5g-mobile-platform, 2019.

[3] Edge tpu. https://github.com/XiaoMi/mace, 2021.

[4] Amazon luna. https://www.amazon.com/luna/landing-page., 2022.

[5] Cloud gaming, meet facebook gaming. https://www.facebook.com/fbgaminghome/blog/cloud-gaming-meetfacebook-gaming., 2022.

[6] Geforce now. https://www.nvidia.com/en-us/geforce-now/., 2022.

[7] Google stadia. https://stadia.google.com/., 2022.

[8] Greedy stays ahead. http://www.cs.cornell.edu/courses/cs482/2003su/handouts/greedy_ahead.pdf., 2022.

[9] How apple personalizes siri without hoovering up your data. https://www.technologyreview.com/2019/12/11/131629/apple-ai-personalizes-sirifederated-learning/, 2022.

[10] Massively scale your deep learning training with nccl 2.4. https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/, 2022.

[11] Optimize training performance with reduction server on vertex ai. https://cloud.google.com/blog/topics/developers-practitioners/optimize-training-performance-reduction-server-vertex-ai, 2022.

[12] Smart camera. https://www.qualcomm.com/products/technology/processors/application-processors/qcs603, 2022.

[13] X-cloud game pass. https://www.xbox.com/en-US/xbox-game-pass/cloud-gaming?xr=shellnav., 2022.

[14] Soc-cluster. https://www.vclusters.com/productinfo1.html, 2023.

[15] Soc-cluster. https://ai-benchmark.com/ranking.html, 2023.

[16] Soc-cluster. https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-1-mobile-platform, 2023.

[17] Soc-cluster. https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-2-mobile-platform, 2023.

[18] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[19] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 472–487, 2022.

[20] Youhui Bai, Cheng Li, Quan Zhou, Jun Yi, Ping Gong, Feng Yan, Ruichuan Chen, and Yinlong Xu. Gradient compression supercharged high-performance data parallel dnn training. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 359–375, 2021.

[21] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. *Advances in neural information processing systems*, 31, 2018.

[22] Amotz Bar-Noy and Guy Kortsarz. Minimum color sum of bipartite graphs. *Journal of Algorithms*, 28(2):339–365, 1998.

[23] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, 52(4):1–43, 2019.

[24] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388, 2019.

[25] Dongqi Cai, Qipeng Wang, Yuanqiang Liu, Yunxin Liu, Shangguang Wang, and Mengwei Xu. Towards ubiquitous learning: A first measurement of on-device training performance. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, pages 31–36, 2021.

[26] Dongqi Cai, Yaozong Wu, Shangguang Wang, Felix Xiaozhu Lin, and Mengwei Xu. Autofednlp: An efficient fednlp framework. *arXiv preprint arXiv:2205.10162*, 2022.

[27] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[28] Varun Chandrasekaran, Suman Banerjee, Diego Perino, and Nicolas Kourtellis. Hierarchical federated learning with privacy. *arXiv preprint arXiv:2206.05209*, 2022.

[29] Xi Chen, Xiaolin Hu, Hucheng Zhou, and Ningyi Xu. Fxpnet: Training a deep convolutional neural network in fixed-point representation. In *2017 International Joint Conference on Neural Networks*, pages 2494–2501. IEEE, 2017.

[30] Yangrui Chen, Yanghua Peng, Yixin Bao, Chuan Wu, Yibo Zhu, and Chuanxiong Guo. Elastic parameter server load distribution in deep learning clusters. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 507–521, 2020.

[31] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[32] Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

[33] Anish Das, Young D Kwon, Jagmohan Chauhan, and Cecilia Mascolo. Enabling on-device smartphone gpu based training: Lessons learned. In *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events*, pages 533–538. IEEE, 2022.

[34] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker,

[1] https://zenodo.org/doi/10.5281/zenodo.8406266

Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[35] Nikoli Dryden, Naoya Maruyama, Tim Moon, Tom Benson, Andy Yoo, Marc Snir, and Brian Van Essen. Aluminum: An asynchronous, gpu-aware communication library optimized for large-scale training of deep neural networks on hpc systems. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2018.

[36] Anis Elgabli, Jihong Park, Amrit S Bedi, Mehdi Bennis, and Vaneet Aggarwal. Gadmm: Fast and communication efficient framework for distributed machine learning. *J. Mach. Learn. Res.*, 21(76):1–39, 2020.

[37] Petko Georgiev, Nicholas D Lane, Kiran K Rachuri, and Cecilia Mascolo. Dsp. ear: Leveraging co-processor support for continuous audio sensing on smartphones. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 295–309, 2014.

[38] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[39] Jialiang Han, Yun Ma, Qiaozhu Mei, and Xuanzhe Liu. Deeprec: On-device deep learning for privacy-preserving sequential recommendation in mobile commerce. In *Proceedings of the Web Conference 2021*, pages 900–911, 2021.

[40] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[41] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy Campbell. Tictac: Accelerating distributed deep learning with communication scheduling. *Proceedings of Machine Learning and Systems*, 1:418–430, 2019.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[43] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. *Advances in neural information processing systems*, 26, 2013.

[44] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, 30, 2017.

[45] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[46] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[47] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 82–95, 2017.

[48] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[49] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 402–416, 2022.

[50] KR Jayaram, Vinod Muthusamy, Gegi Thomas, Ashish Verma, and Mark Purcell. Adaptive aggregation for federated learning. *arXiv preprint arXiv:2203.12163*, 2022.

[51] Fucheng Jia, Deyu Zhang, Ting Cao, Shiqi Jiang, Yunxin Liu, Ju Ren, and Yaoxue Zhang. Codl: efficient cpu-gpu co-execution for deep learning inference on mobile devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pages 209–221. Association for Computing Machinery New York, NY, USA, 2022.

[52] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

[53] Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yu Cai, Tianhang Yu, et al. Mnn: A universal and efficient inference engine. *arXiv preprint arXiv:2002.12418*, 2020.

[54] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation*, pages 463–479, 2020.

[55] Cinar Kilcioglu, Justin M Rao, Aadharsh Kannan, and R Preston McAfee. Usage patterns and the economics of the public cloud. In *Proceedings of the 26th International Conference on World Wide Web*, pages 83–91, 2017.

[56] Youngsok Kim, Joonsung Kim, Dongju Chae, Daehyun Kim, and Jangwoo Kim. μlayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–15, 2019.

[57] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[58] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[60] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 1–12. IEEE, 2016.

[61] Seyyed Salar Latifi Oskouei, Hossein Golestani, Matin Hashemi, and Soheil Ghiasi. Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 1201–1205, 2016.

[62] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.

[63] Mu Li. *Scaling distributed machine learning with system and algorithm co-design*. PhD thesis, PhD thesis, Intel, 2017.

[64] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27, 2014.

[65] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[66] Wenjie Li, Qiaolin Xia, Hao Cheng, Kouyin Xue, and Shu-Tao Xia. Vertical semi-federated learning for efficient online advertising. *arXiv*

preprint arXiv:2209.15635, 2022.

[67] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

[68] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *Advances in neural information processing systems*, 30, 2017.

[69] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep Gradient Compression: Reducing the communication bandwidth for distributed training. In *The International Conference on Learning Representations*, 2018.

[70] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*, 2015.

[71] TensorFlow Lite. Sharing a gpu between mpi processes: multiprocess service(mps). https://docs.nvidia.com/deploy/mps/index.html, 2019.

[72] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision*, December 2015.

[73] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[74] Naram Mhaisen, Alaa Awad Abdellatif, Amr Mohamed, Aiman Erbad, and Mohsen Guizani. Optimal user-edge assignment in hierarchical federated learning based on statistical properties and network topology constraints. *IEEE Transactions on Network Science and Engineering*, 9(1):55–66, 2021.

[75] Byunggook Na, Jaehee Jang, Seongsik Park, Seijoon Kim, Joonoo Kim, Moon Sik Jeong, Kwang Choon Kim, Seon Heo, Yoonsang Kim, and Sungroh Yoon. Scalable smartphone cluster for deep learning. *arXiv preprint arXiv:2110.12172*, 2021.

[76] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.

[77] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. Billion-scale federated learning on mobile clients: A submodel design with tunable privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[78] Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Gpt3-to-plan: Extracting plans from text using gpt-3. *arXiv preprint arXiv:2106.07131*, 2021.

[79] Panos M Pardalos, Thelma Mavridou, and Jue Xue. The graph coloring problem: A bibliographic survey. In *Handbook of combinatorial optimization*, pages 1077–1141. Springer, 1998.

[80] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[81] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[82] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in neural information processing systems*, 24, 2011.

[83] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. Zero-offload: Democratizing billion-scale model training. In *2021 USENIX Annual Technical Conference*, pages 551–564, 2021.

[84] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtárik. Scaling distributed machine learning with in-network aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation*, pages 785–808, 2021.

[85] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[86] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[87] Jaeyong Song, Jinkyu Yim, Jaewon Jung, Hongsun Jang, Hyung-Jin Kim, Youngsok Kim, and Jinho Lee. Optimus-cc: Efficient large nlp model training with 3d parallelism aware communication compression. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 560–573, 2023.

[88] Jennifer Switzer, Gabriel Marcano, Ryan Kastner, and Pat Pannuto. Junkyard computing: Repurposing discarded smartphones to minimize carbon. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 400–412, 2023.

[89] Olivier Valery, Pangfeng Liu, and Jan-Jan Wu. A collaborative cpu-gpu approach for deep learning on mobile devices. *Concurrency and Computation: Practice and Experience*, 31(17):e5225, 2019.

[90] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[91] Omar Abdel Wahab, Gaith Rjoub, Jamal Bentahar, and Robin Cohen. Federated against the cold: A trust-based federated learning approach to counter the cold start problem in recommendation systems. *Information Sciences*, 601:189–206, 2022.

[92] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Nikhil Devanur, Jorgen Thelin, and Ion Stoica. Blink: Fast and generic collectives for distributed ml. *Proceedings of Machine Learning and Systems*, 2:172–186, 2020.

[93] Jianyu Wang and Gauri Joshi. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd. *Proceedings of Machine Learning and Systems*, 1:212–229, 2019.

[94] Maolin Wang, Seyedramin Rasoulinezhad, Philip HW Leong, and Hayden K-H So. Niti: Training integer neural networks using integer-only arithmetic. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):3249–3261, 2022.

[95] Qipeng Wang, Mengwei Xu, Chao Jin, Xinran Dong, Jinliang Yuan, Xin Jin, Gang Huang, Yunxin Liu, and Xuanzhe Liu. Melon: Breaking the memory wall for resource-efficient on-device machine learning. 2022.

[96] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[97] Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen Jarvis. Safa: A semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Transactions on Computers*, 70(5):655–668, 2020.

[98] Xundong Wu, Yong Wu, and Yong Zhao. Binarized neural networks on the imagenet classification task. *arXiv preprint arXiv:1604.03058*, 2016.

[99] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[100] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. Antman: Dynamic scaling on gpu clusters for deep learning. In *14th USENIX Symposium on*

Operating Systems Design and Implementation, pages 533–548, 2020.

[101] Daliang Xu, Mengwei Xu, Qipeng Wang, Shangguang Wang, Yun Ma, Kang Huang, Gang Huang, Xin Jin, and Xuanzhe Liu. Mandheling: mixed-precision on-device dnn training with dsp offloading. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pages 214–227, 2022.

[102] Mengwei Xu, Zhe Fu, Xiao Ma, Li Zhang, Yanan Li, Feng Qian, Shangguang Wang, Ke Li, Jingyu Yang, and Xuanzhe Liu. From cloud to edge: a first look at public edge platforms. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 37–53, 2021.

[103] Mengwei Xu, Feng Qian, Qiaozhu Mei, Kang Huang, and Xuanzhe Liu. Deeptype: On-device deep learning for input personalization service with minimal privacy concern. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–26, 2018.

[104] Mengwei Xu, Li Zhang, and Shangguang Wang. Position paper: Renovating edge servers with arm socs. In *2022 IEEE/ACM 7th Symposium on Edge Computing*, pages 216–223. IEEE Computer Society, 2022.

[105] Xiang Yang. Shuffle-exchange brings faster: Reduce the idle time during communication for decentralized neural network training. *arXiv preprint arXiv:2007.00433*, 2020.

[106] Yang You, Yuhui Wang, Huan Zhang, Zhao Zhang, James Demmel, and Cho-Jui Hsieh. The limit of the batch size. *arXiv preprint arXiv:2006.08517*, 2020.

[107] Jinliang Yuan, Mengwei Xu, Xiao Ma, Ao Zhou, Xuanzhe Liu, and Shangguang Wang. Hierarchical federated learning through lan-wan orchestration. *arXiv preprint arXiv:2010.11612*, 2020.

[108] Quan Yuan, Haibo Zhou, Jinglin Li, Zhihan Liu, Fangchun Yang, and Xuemin Sherman Shen. Toward efficient content delivery for automated driving services: An edge computing solution. *IEEE Network*, 32(1):80–86, 2018.

[109] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. Poseidon: An efficient communication architecture for distributed deep learning on gpu clusters. In *2017 USENIX Annual Technical Conference*, pages 181–193, 2017.

[110] Li Zhang, Zhe Fu, Boqing Shi, Xiang Li, Rujin Lai, Chenyang Chen, Ao Zhou, Xiao Ma, Shangguang Wang, and Mengwei Xu. Soc-cluster as an edge server: an application-driven measurement study. *arXiv preprint arXiv:2212.12842*, 2022.

[111] Qiyang Zhang, Xiang Li, Xiangying Che, Xiao Ma, Ao Zhou, Mengwei Xu, Shangguang Wang, Yun Ma, and Xuanzhe Liu. A comprehensive benchmark of deep learning libraries on mobile devices. In *Proceedings of the ACM Web Conference 2022*, pages 3298–3307, 2022.

[112] Wei Zhang, Binghao Chen, Zhenhua Han, Quan Chen, Peng Cheng, Fan Yang, Ran Shu, Yuqing Yang, and Minyi Guo. Pilotfish: Harvesting free cycles of cloud gaming with deep learning training. In *2022 USENIX Annual Technical Conference*, pages 217–232, 2022.

[113] Xin Zhang, Jia Liu, Zhengyuan Zhu, and Elizabeth S Bentley. Compressed distributed gradient descent: Communication-efficient consensus over networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2431–2439. IEEE, 2019.

[114] Xishan Zhang, Shaoli Liu, Rui Zhang, Chang Liu, Di Huang, Shiyi Zhou, Jiaming Guo, Qi Guo, Zidong Du, Tian Zhi, et al. Fixed-point back-propagation training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2330–2338, 2020.

[115] Zhenxiao Zhang, Zhidong Gao, Yuanxiong Guo, and Yanmin Gong. Scalable and low-latency federated learning with cooperative mobile edge networking. *arXiv preprint arXiv:2205.13054*, 2022.

[116] Kang Zhao, Sida Huang, Pan Pan, Yinghan Li, Yingya Zhang, Zhenyu Gu, and Yinghui Xu. Distribution adaptive int8 quantization for training cnns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3483–3491, 2021.

[117] Xing Zhao, Aijun An, Junfeng Liu, and Bao Xin Chen. Dynamic stale synchronous parallel distributed training for deep learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1507–1517. IEEE, 2019.

[118] Qihua Zhou, Song Guo, Zhihao Qu, Jingcai Guo, Zhenda Xu, Jiewei Zhang, Tao Guo, Boyuan Luo, and Jingren Zhou. Octo:{INT8} training with loss-aware compensation and backward quantization for tiny on-device learning. In *2021 USENIX Annual Technical Conference*, pages 177–191, 2021.

[119] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[120] Yi Zhou, Yaoliang Yu, Wei Dai, Yingbin Liang, and Eric Xing. On convergence of model parallel proximal gradient algorithm for stale synchronous parallel system. In *Artificial Intelligence and Statistics*, pages 713–722. PMLR, 2016.

[121] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. Adaptive quantization for deep neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[122] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[123] Feng Zhu, Ruihao Gong, Fengwei Yu, Xianglong Liu, Yanfei Wang, Zhelong Li, Xiuqi Yang, and Junjie Yan. Towards unified int8 training for convolutional neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1969–1979, 2020.