Efficient, Scalable, and Sustainable DNN Training on SoC-Clustered Edge Servers

Mengwei Xu *Member, IEEE*, Daliang Xu, Chiheng Lou, Li Zhang, Gang Huang *Senior Member, IEEE*, Xin Jin *Senior Member, IEEE*, Xuanzhe Liu *Senior Member, IEEE*

Abstract— In the realm of industrial edge computing, a novel server architecture known as SoC-Cluster, characterized by its aggregation of numerous mobile systems-on-chips (SoCs), has emerged as a promising solution owing to its enhanced energy efficiency and seamless integration with prevalent mobile applications. Despite its advantages, the utilization of SoC-Cluster servers remains unsatisfactory, primarily attributed to the tidal patterns of user-initiated workloads. To address such inefficiency, we introduce `SoCFlow+`, a pioneering framework designed to facilitate the co-location of deep learning training tasks on SoC-Cluster servers, thereby optimizing resource utilization.

`SoCFlow+` incorporates three novel techniques tailored to mitigate the inherent limitations of commercial SoC-Cluster servers. First, it employs group-wise parallelism complemented by delayed aggregation, a strategy engineered to enhance the training efficiency and scalability of deep learning models, effectively circumventing network bottlenecks. Second, it integrates a data-parallel mixed-precision training algorithm, optimized to exploit the heterogeneous processing capabilities inherent to mobile SoCs fully. Third, `SoCFlow+` employs an underclocking-aware workload re-balanacing mechanism to tackle the training performance degradation caused by the thermal control of mobile SoCs. Through rigorous experimental validation, `SoCFlow+` achieves a convergence speedup ranging from 1.6× to 740× across 32 SoCs, compared to conventional benchmarks. Furthermore, when juxtaposed with commodity GPU servers (e.g., NVIDIA V100) under identical power constraints, `SoCFlow+` not only exhibits comparable training speed but also achieves a remarkable reduction in energy consumption by a factor of 2.31× to 10.23×, all while preserving convergence accuracy.

Index Terms—Edge computing, distributed training

1 Introduction

The burgeoning field of edge computing has witnessed the advent of a novel server architecture, termed SoC-Cluster, which is predicated on the integration of a multitude of mobile system-on-chips (SoCs) [1], [2]. This innovation distinguishes itself from conventional cloud or edge datacenter servers by offering a more compact computing solution, boasting an ability to house up to 480 physical cores within a 2U rack space through the deployment of 60 Snapdragon 865 SoCs. Such a configuration not only promises superior energy efficiency but also facilitates the seamless execution of native mobile applications without necessitating modifications. Consequently, SoC-Clusters have found extensive applications in edge clouds, serving a myriad of mobile applications ranging from cloud gaming [3], [4] to live streaming [5], underscored by their deployment in millions of units globally.

Despite their widespread adoption, a meticulous examination of SoC-Cluster server utilization in §2 reveals a paradox of under-utilization. Data obtained from thousands of such servers in actual deployment scenarios unveil that over 95% of these SoCs operate below 20% CPU usage on average. This under-utilization is primarily attributed to the user-driven nature of the hosted applications, which inherently exhibit significant fluctuations in resource demand, manifesting a tidal usage pattern. This observation is corroborated by prior studies [6] and highlights an inefficiency in resource allocation that merits addresal.

In light of this under-utilization, we propose the co-location of deep learning (DL) training tasks on idle SoCs within SoC-Clusters. This proposition is driven by three pivotal reasons. First, the edge cloud represents a nexus where mobile user data accumulates, making it an ideal venue for DL model training [7], [8], [9], [10], [11], [12], [13]. This approach not only alleviates backbone network load by minimizing data transit distances but also addresses privacy concerns through local data consumption. Moreover, it enables geographically tailored model training, enhancing personalization in applications such as item recommendations [14], [15], [16]. Second, DL training tasks, characterized by their predictable and delay-tolerant nature, complement existing workloads on SoC-Clusters, offering a practical solution to bridge the utilization gap. Finally, the burgeoning research in mobile SoCs equipped with on-chip accelerators has significantly advanced the feasibility of executing DL tasks on such platforms, promising enhanced computational efficiency and energy savings [17], [18], [19].

However, the venture into DL training on SoC-Clusters is not devoid of challenges. Our initial forays into this domain in §2.2 revealed that singular SoCs fall short of the computational prowess required for timely training of

- *Mengwei Xu and Li Zhang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China.*
  E-mail: {mwx;li.zhang}@bupt.edu.cn.
- *Daliang Xu, Chiheng Lou, Gang Huang, Xin Jin, and Xuanzhe Liu are with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education; School of Computer Science, Peking University, Beijing, China*
  E-mail: {xudaliang;hg;xinjinpku;liuxuanzhe}@pku.edu.cn, louchiheng23@stu.pku.edu.cn.

Manuscript received March 30, 2024.

medium-sized models. For instance, training VGG-11 [20] 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. Such 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. This predicament nudges us towards distributed training across multiple SoCs as a viable acceleration mechanism. 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 [21], [22], [23], we find it introduces its own set of hurdles, which collectively necessitate innovative solutions to harness the full potential of SoC-Clusters for DL training.

• *Limited network bandwidth.* Datacenters traditionally offer robust network infrastructures, providing up to 100Gbps bandwidth to accommodate the demands of distributed training workloads [23]. In stark contrast, the network within a typical SoC-Cluster server is constrained to a mere 1Gbps. This limitation is further exacerbated by the shared nature of the network among multiple SoCs, a consequence of the centralized network switch design inherent to the SoC architecture. Such a configuration, while sufficient for less demanding applications like cloud gaming, becomes a significant bottleneck when subjected to the intensive network traffic of distributed DNN training. This bottleneck effect hinders the scalability of traditional datacenter-oriented network topologies, such as the Parameter Server [21] and Ring-AllReduce [24], within the SoC-Cluster environment.

• *Heterogeneous processors with mixed data formats.* The diversity of processors within mobile SoCs, including domain-specific accelerators, presents a unique set of challenges for DL workload optimization. Specifically, (i) mobile GPUs have been shown to be suboptimal for training tasks [25], [26], [27]; and (ii) mobile NPUs, while capable of accelerating training, necessitate low-precision formats such as INT8, which can lead to trade-offs between training speed and model accuracy [12]. Addressing this duality requires the development of sophisticated mixed-precision training algorithms and aggregation schemes that can harmonize the disparate computational capabilities and data formats of mobile CPUs and NPUs to achieve both efficient training and satisfactory model accuracy.

• *Hardware underclocking.* Edge servers are often deployed on harsh environments where dedicated cooling mechanisms are not available, e.g., cellular base stations, retail stores, outdoor kiosks/ATMs, etc [28]. Mobile SoCs are inherently less tolerant to thermal stress compared to their datacenter counterparts, a characteristic that poses significant challenges during sustained distributed training sessions. The accumulation of heat within the SoC-Cluster can trigger protective underclocking mechanisms in SoCs, a phenomenon observed during the training of ResNet-18 on CIFAR-10, where instances of underclocking were noted, each lasting from several seconds to minutes [29]. Such thermal-induced underclocking disrupts the balance of workload distribution across SoCs, leading to inefficiencies and delays in the training process.

This paper introduces `SoCFlow+`, a pioneering framework designed to surmount these challenges and facilitate efficient DL model training on SoC-Cluster servers. At its core, `SoCFlow+` aims to amplify training speeds in proportion to the number of participating SoCs, focusing on models of sizes and complexities pertinent to edge cloud requirements. It achieves this through the introduction of following novel techniques.

Group-wise parallelism with delayed aggregation (§3.1). Inspired by the prior efforts on communication-efficient distributed training [30], [31] and federated learning protocols [8], `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.

Passive training-cooling co-design (§3.3) We conduct an exhaustive examination of the interplay between temperature dynamics and fan speed control, revealing that the energy demand for cooling operations to prevent SoC underclocking parallels that of ten fully engaged SoCs. In light of these findings, `SoCFlow+` introduces a novel passive training-cooling co-design aimed at optimizing fan energy consumption while accommodating minor underclocking instances. This approach strategically minimizes the impact on energy efficiency. Additionally, `SoCFlow+` implements a workload redistribution mechanism to address and mitigate the implications of underclocking, thereby ensuring a marginal effect on overall system performance.

We have fully implemented `SoCFlow+` on top of MNN [32], the state-of-the-art training library on mobile SoCs. `SoCFlow+` supports models exported from TensorFlow [33] and PyTorch [22]. 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 [34], [35], [36], [37], [23], [24], [38], including traditional Parameter Server and

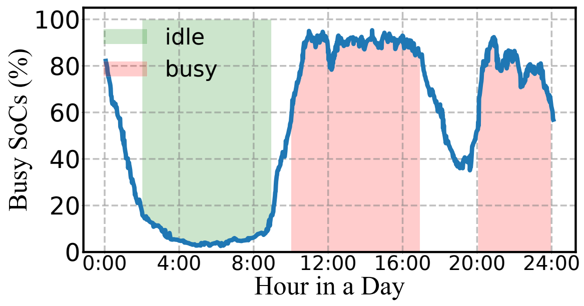


Fig. 1. Busy SoCs ratio within a day on deployed SoC-Cluster servers.

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

This work targets SoC-Cluster, a unique form of edge server that consists of massive, low-power ARM-based system-on-chips (SoCs). For more background and measurements of this platform, please refer to our previous work [39], [2].

2.1 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

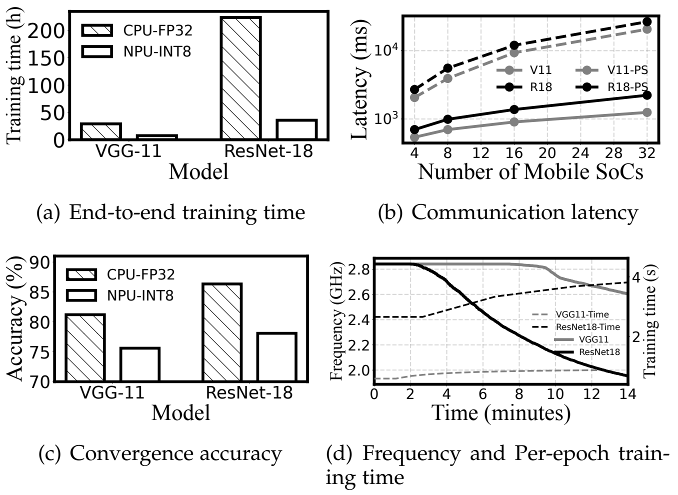


Fig. 2. Measurement results of training VGG-11 (V11) and ResNet-18 (R18) models on CIFAR-10 dataset atop edge SoC cluster.

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 1. This phenomenon attributes to the inherent, user-centric characteristics of the workloads hosted on SoC-Clusters. It is consistent with the recent large-scale empirical study on commercial edge platforms [6], [39].

To increase the hardware utilization, a typical method is to co-locate best-effort workloads with those latency-critical workloads on servers [40], [41], [42]. In this work, we seek to perform DNN training [42] 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 experimentally shown in §4.

2.2 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. 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 [20] and ResNet-18 [29] on CIFAR-10 dataset [43]) atop the SoC-Cluster presented above, using the state-of-the-art mobile training engines `MNN` [32]. 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 [12]? The results are illustrated in Figure 2.

• **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 2(a). Even with state-of-the-art mixed-precision training algorithms [44], 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 2(b). In our experiments, a PCB board contains 5 SoCs. 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 [45], [46], [47], [48], 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 2(c).

• **Observation #4: The training efficiency suffers from the high SoC temperature and under-clocked SoCs.** Unlike hardware designed for clouds, SoCs have tighter constraints for its operational environments. As Figure 2(d) shows, after 15-minute training, the CPU frequency drops from 2.8GHz to 1.8GHz or 2.55GHz as the SoC temperature rises sharply from 28°C to 46°C. Such a phenomenon results in up to 59–76% per-epoch training delays, as shown in Figure 2(d).

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 3(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:

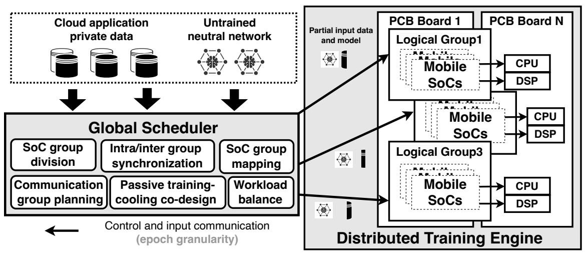
• **Global scheduler** is a lightweight software deployed on the SoC-Cluster's control board. Its primary function is to coordinate the training process. Ahead of the training, it determines how SoCs will be orchestrated, such as SoC grouping, inter-/intra-group gradients synchronizing frequency, and 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. Throughout the training process, the SoC-Cluster may incur underclocking issues that could delay the training. The scheduler also deals with the issues by passive training-cooling co-design (§3.3). 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.

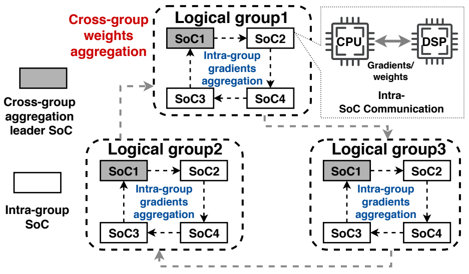
3.1 Group-wise Parallelism with Delayed Aggregation

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

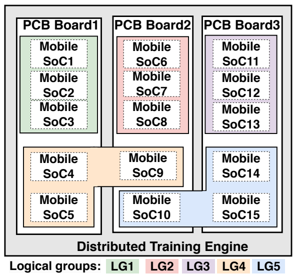
- One is to design an efficient network topology, which specifies how data (model weight updates in our case) flows and aggregates across SoCs. In data centers, Ring-AllReduce [24] is a bandwidth-optimal communication strategy and is widely used in network-constrained scenarios. However, it is still inadequate for SoC-Cluster, as previously shown in Figure 2. 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 [49], [37], where the clients are geo-distributed and are connected to a central server through a wireless network. For instance, FedAvg [37] 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 bottleneck is mitigated. This approach, however, causes model staleness and potential accuracy degradation [37].


Cloud application private data
Untrained neutral network
Global Scheduler
SoC group division
Intra/inter group synchronization
SoC group mapping
Communication group planning
Passive training-cooling co-design
Workload balance
Control and input communication
(epoch granularity)
Partial input data and model
PCB Board 1
PCB Board N
Logical Group1
Mobile SoCs
CPU
DSP
Logical Group3
Distributed Training Engine

(a) A simplified workflow.



(b) Communication topology.



(c) Logical-to-physical mapping.

Fig. 3. The overview of `SoCFlow+`.

TABLE 1
The symbols and terms used in §3

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.

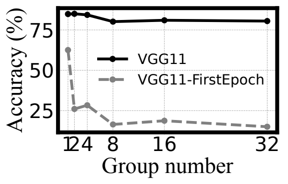
The network capacity of SoC-Cluster lies somewhere between the high-speed data center and wireless network. Therefore, we propose a hierarchical topology that enables group-wise parallelism across SoCs, as shown in Figure 3(b). `SoCFlow+` divides the SoCs into *logical groups*: (1) Within a logical group, 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) [50], [51], `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 3(b)) to aggregate weights with other groups, and all groups' leaders also form a Ring-AllReduce topology.

There are four crucial steps to efficiently realize the proposed mechanism: (1) determining the number of SoCs of each logical group, i.e., the group size; (2) controlling the inter-group communication frequency; (3) mapping the logical topology of SoC groups into the physical SoC-Cluster architecture; (4) planning the group-wise communication to minimize the NIC contention during training. The terms used in this section are shown in Table 1.

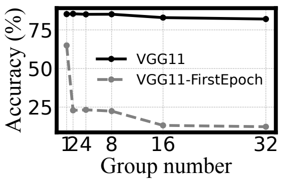
Determining group size. Supposing M SoCs will be divided into N groups, and each group's global batch size is BS_g, the per-epoch training time can be formulated as

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

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 [52], [53], [54]. This is also confirmed by our experiments in Figure 4, which shows a too large group number notably degrades convergence accuracy.



(a) VGG11



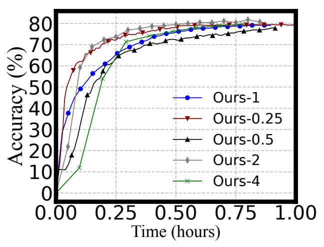
(b) ResNet18

Fig. 4. The testing accuracy after achieving final convergence and for only the first epoch is compared across different group sizes.

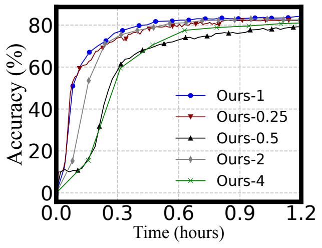
To identify the largest group size N that guarantees minimal training time while preserving convergence accuracy, `SoCFlow+` offers system designers the flexibility to empirically select this parameter by default and provides an optional heuristic approach. This approach capitalizes on an observation: the training accuracy observed during the initial epoch closely mirrors the behavior of convergence accuracy, as shown in Figure 4. 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 4(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.

Controlling inter-group communication frequency. Inter-group communication frequency is a crucial factor influ-



(a) VGG-11



(b) ResNet-18

Fig. 5. Convergence accuracy across clock time under different inter-group communication frequency. Ours-X represents `SoCFlow+` uses X epochs as the inter-group communication interval.

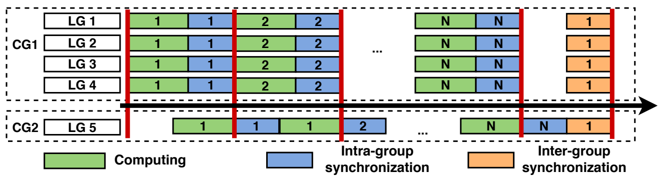


Fig. 6. The group-wise communication planning used by `SoCFlow+`. *CG* and *LG* represent the communication group and logical group, respectively.

encing both convergence speed and convergence accuracy. Typically, a higher communication frequency leads to converge faster but with lower accuracy, since frequent synchronization increases the model's learning ability but is more likely to get into the local optimum. Our pilot experiments support this observation. We comprehensively investigate the end-to-end training performance of `SoCFlow+` under 32 SoCs with communication frequencies of 0.25, 0.5, 1, 2, and 4 epochs, as shown in Figure 5. According to the two models' results, Ours-0.25 always converges the fastest but with lower accuracy. Ours-1/Ours-2 have the highest convergence accuracy, e.g., 82.83% and 84.35% for VGG-11 and ResNet-18, respectively.

To achieve both higher convergence accuracy and faster convergence speed, `SoCFlow+` reduces inter-group communication frequency in a manner similar to learning rate decay [55], [56]. We initialize the communication frequency at a value of 0.25. Following every GN epochs, the communication frequency undergoes a half decay process until it reaches the minimum communication frequency value (e.g., 2 in the majority of cases). The default value of GN in `SoCFlow+` is 10, and `SoCFlow+` allows user sets its value empirically.

Mapping the logical topology into the physical SoC-Cluster architecture. 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 3(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.

We have the following theorems for the *integrity-greedy mapping* algorithm. The first theorem can be proven by the "greedy stays ahead" algorithm [57]. 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 3(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 6. 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 [58], 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) [59].

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 [59] 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 6 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 §2.2, 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 3(b). Currently, we employ the standard SGD as the training optimizer on CPU and the state-of-the-art INT8-based optimizer [44] 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 [60], [61], [36], [62], 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 [63]. 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.

3.3 Passive Training-cooling Co-design

Temperature control is often a pressing issue for edge server deployments [28], especially when there is no dedicated cooling facility, e.g., cellular base stations, retail stores, outdoor kiosks/ATMs, etc. Specifically, SoC-Cluster leverages air cooling strategy (e.g., fans) to radiating devices; such strategy is easier to use and cheaper than free or water

TABLE 2
A summary of fan speed and corresponding SoC temperature, performance decline, and energy overhead for VGG-11 model.

Speed	Power	Temp.	Perf. decline	Power overhead
1100	0 W	39°C	30%	0%
3250	3 W	39°C	30%	1.5%
5700	9 W	39°C	30%	4.5%
8300	25 W	34–35°C	5%	12.5%
11000	51 W	32°C	0%	25.5%
14000	107 W	31°C	0%	53.5%

cooling [28], [64], [65]. However, air cooling struggles to satisfy the strict cooling demand at SoC-Cluster's high power density due to its low heat conduction capacity and the difficulty in managing the airflow efficiently when the rack and aisle are increasingly compact [28]. Worse, since the temperature of the SoCs highly depends on the environment temperature, hardware specifications, and co-located workloads, some specific SoCs will underclock unavoidably.

To mitigate the impacts of underclocking, one straightforward method is to avoid it by increasing the fan speed during training, i.e., an *active* manner. To this end, we study the relation between the fan speed and the SoC temperature on a typical SoC-Cluster with 60 Snapdragon 865 SoCs. The experiments are performed on 40 of the SoCs under ambient temperature (25°C). Note that the underclocking is triggered by the SoC hardware, instead of the software.

As shown in Table 2, increasing the fan speed over 11,000 can maintain the temperature around 32°C without underclocking or performance degradation. However, it incurs 51 watts of power consumption overhead, which equals 10 SoCs running at full utilization. Instead, one might use a fan speed of 8,300 that incurs relatively lower energy overhead (25 watts) with an acceptable performance loss (5%) due to underclocking. This approach, however, still has two drawbacks. First, its energy consumption is still nontrivial and higher than data center infrastructure with highly optimized cooling facilities. Second, such an active cooling decision is rather ad-hoc: the temperature of the SoCs highly depends on the environment temperature, hardware specifications, workloads, etc. Note that the SoCs of a single SoC-Cluster might serve different types of workloads simultaneously. It makes an active method that always obtains an "oracle" fan speed through profiling impractical.

Passive cooling strategy. Instead, `SoCFlow+` uses a more aggressive and passive energy management strategy: in most of the training time, `SoCFlow+` uses a relatively low fan speed (i.e., 3,250 in the above case) to reduce energy consumption; when the SoCs experience underclocking, `SoCFlow+` boosts the fan speed to cool down the devices. Such a passive strategy makes `SoCFlow+` applicable to ubiquitous scenarios, disregarding the underlying hardware and workloads. Furthermore, it is even more energy efficient than using a static, "oracle" fan speed. The rationale is based on a key observation: it takes much time (tens of minutes) for the SoC-Cluster to go overheated from a relatively low temperature. As shown in Figure 7 (green line), it takes about 10–12 minutes to underclock when training the VGG-11 model on 40 SoCs, while cooling the SoCs only needs 2–3 minutes. Such a phenomenon motivates our passive

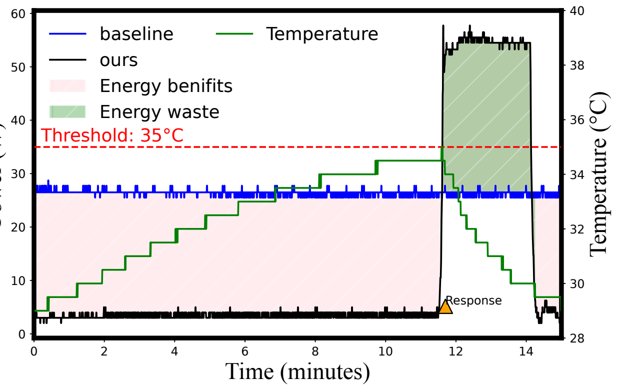


Fig. 7. The benefits of the passive cooling strategy. Notably, the SoC temperature API operates with a minimum granularity of 0.5°C.

training-cooling co-design. `SoCFlow+` sets the the temperature threshold as 35°C to avoid obvious underclocking (5% performance decline). When the SoC temperature rises higher than 35°C, `SoCFlow+` will respond and increase the fan speed to cool the SoC-Cluster.

Underclocking-aware training re-balancing. To further mitigate the training performance degradation caused by the above passive cooling design, `SoCFlow+` re-balances the workloads both within and across SoC groups. To illustrate how underclocking problem affects the training efficiency, we first give per-iteration training time formulation, as follows.

$$T_{iter} = \max\{T_i,\ i \in communication\ group\ \mathcal{J}\} \quad (7)$$

$$T_i = \max\{T_k^{Cal} + T_k^{sync},\ k \in SoC\ group\ \mathcal{G}_i\} \quad (8)$$

where T_k^{Cal}, T_k^{sync}, and T_i are SoC k's calculation and synchronization time, and SoC group G_i's training time. Typically, underclocking does not influence synchronization since synchronization are mainly bound to network bandwidth. To that end, the two most important factors are T_k^{Cal}, SoC's calculation time within a group, and T_i, training time across SoC groups.

• *Within a group*: `SoCFlow+` re-balances workloads into varying size portions by input data according to SoCs' compute capacity. Supposing SoCs' capacity are $\{C_1, C_2, \cdots, C_n\}$ and the total input data is $\mathcal{D}$, the input data that each SoC should train is

$$D_i = \mathcal{D} * \frac{C_i}{\sum_{k=1}^{n} C_k} \quad (9)$$

Correspondingly, the gradients aggregation and parameter update should change simultaneously according to input data size.

$$W_{t+1} = W_t - Opt(\frac{1}{n} * \sum_{i=1}^{n}(g_i * \frac{C_i}{\sum_{k=1}^{n} C_k})) \quad (10)$$

where W_{t+1}, W_t and Opt represents the t+1 iteration weights, t iteration weights, and optimizer function, respectively.

• *Across groups*: Since different SoC groups' weight aggregation is epoch grained, naively balancing workloads will lead to accuracy degradation. Thus, `SoCFlow+` re-balances compute capacity for each SoC group rather than

TABLE 3
DNN models used in the experiments.

Model	Dataset	Learning methods
LeNet [67]	EMNIST [68]	From scratch
	Fashion-MNIST [69]	
VGG-11 [20]	CIFAR-10 [43]	
	CelebA [70]	
ResNet-18 [29]	CIFAR-10	
	CelebA	
MobileNet_V1 [71]	CIFAR-10	
ResNet-50 [29]	CINIC-10 [72]	Transfer learning

workloads. Our observation is that SoCs in one physical group are connected to the NIC through star topology [66], thus homogeneous and position-independent in network communication but with heterogeneous underclocking performance. That means supposing 5 SoCs S_1, S_2, S_3, S_4, S_5 in one physical group, an SoC group $\{S_1, S_2, S_3, S_4\}$ is equivalent to $\{S_4, S_3, S_2, S_1\}$, and even $\{S_1, S_2, S_4, S_5\}$ when only considering network communication. To that end, reordering the SoC sequence in the physical group will not break the *sequential mapping*'s optimal property but can mitigate SoC groups' underclocking heterogeneity.

• *Implementation*: After each epoch's training, each SoC records its total calculation time and sends them to the *global scheduler*. The global scheduler analyzes each SoC's status and generates the corresponding workload partition and SoC's sequence of one physical group to re-balance the training workload.

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 [33] and Pytorch [62]. `SoCFlow+` leverages MNN [32] (the most lightweight on-device training framework) as the CPU backend and Mandheling [12] 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. `SoCFlow+` uses a wired network as the SoCs are cross-connected directly through routers using SAS [73] instead of mobile/wireless network. 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. 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 [43], VGG-11 [20], ResNet-18/50 [29], and MobileNet_V1 [71], as listed in Table 3. 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.

Hyper-parameter settings. `SoCFlow+` employs a standard data-parallel training approach. Common hyper-parameters, such as batch size and learning rate, are selected following conventions in the literature [60], [61], [36], [62]. Specifically, the batch size is set to 64, and the learning rate is initialized at 0.01, with a cosine decay schedule applied for VGG-11 and ResNet-18 models. A unique aspect of `SoCFlow+` is the handling of inter-group communication intervals, which is not commonly addressed in existing works. As detailed in §3.1, `SoCFlow+` initializes the communication frequency at 4, corresponding to a communication interval of 0.25 epochs. Additionally, `SoCFlow+` employs a half decay strategy for communication frequency, similar to the learning rate decay method described in previous studies [55], [56]. The decay occurs every 10 epochs until it reaches the minimum communication frequency value, e.g., 0.5 (2 epochs) in the majority of cases.

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 [36]. (2) Ring-AllReduce (`RING`): the traditional FP32-based allreduce training method following the workflow of Horovod [24]. (3) HiPress [34]: a compression-aware gradient synchronization framework for data-parallel DNN training. It uses DGC [74] as the sparsification compression algorithm. (4) 2D parallelism [38] (`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 [23]. We do not compare 3D parallelism [38] as its tensor parallelism is more suitable for large models like GPT3 [75], 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 [37]. (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 [35], [31] with FedAvg algorithm. Both of the baselines have been implemented within the context of independent and identically distributed (IID) settings. To be noted, FL baselines also adopt the parameter server architecture, but aggregate models in a less frequent manner, e.g., per batch vs. per epoch.

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

TABLE 4
A summary of end-to-end training convergence accuracy. ”Acc.”: accuracy; ”Degrad.”: accuracy degradation.

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**

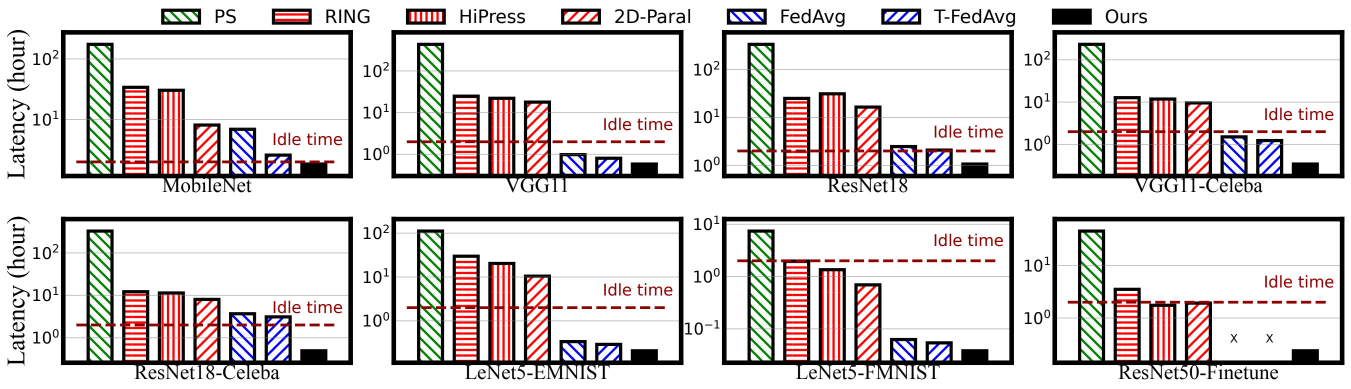


Fig. 8. End-to-end training time up to convergence under different training scenarios.

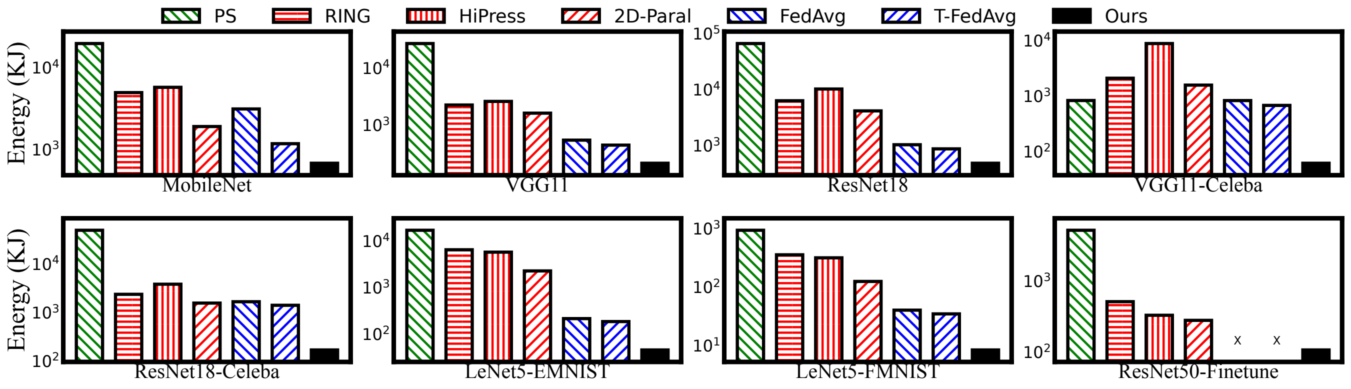


Fig. 9. End-to-end training energy consumption up to convergence under different training scenarios.

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 4, 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 4. 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 communica-

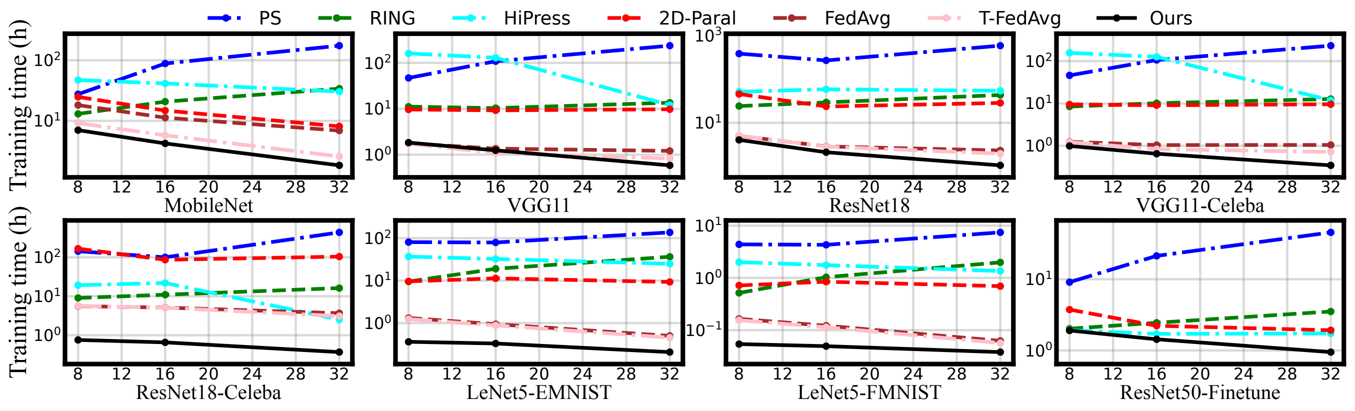


Fig. 10. Elapsed training time taken to reach same target accuracy under various SoC numbers.

tion 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 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, 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. It shows that **SoCFlow+ consistently outperforms all baselines from 8 to 32 SoCs, and its benefits are more prominent with the increasing SoC number.**

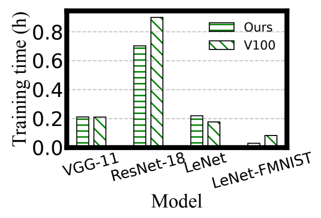
SoCFlow+ reduces training time on 8 SoCs by 83.3×, 8.89×, 2.31×, 36.4×, 2.51×, and 53.8× on average for PS, RING, HiPress, 2D-Paral, FedAvg, and T-FedAvg, respectively; while the speedups for 32 SoCs are 474.8×, 49.3×, 2.35×, 52.8×, 3.1× and 35.7× correspondingly, which are 2.6 × larger than that of 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.

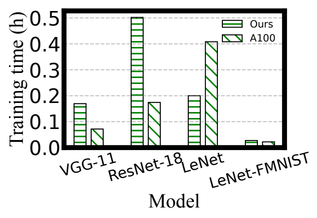
To be noted, SoCFlow+ is mostly designed for DNN training on a single SoC-Cluster server, thereby the scalability analysis is limited to the number of mobile SoCs per server. In the future, we plan to extend SoCFlow+ to span multiple SoC-Clusters to further expand the scalability.

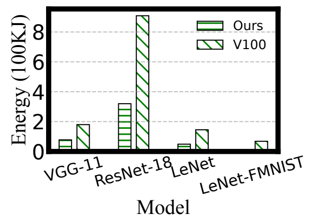
4.4 Comparison with Traditional Datacenter GPU

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 reasons: (1) Relatively consistent performance gap in latest


(a) Training time: 865 v.s. V100


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


(c) Energy: 865 v.s. V100

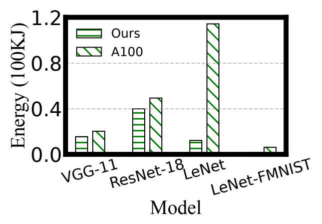
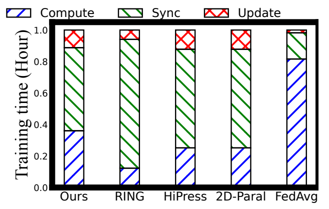

(d) Energy: 8gen1 v.s. A100

Fig. 11. Training time and energy consumption comparison between `SoCFlow+` and traditional datacenter GPUs using PyTorch.


(a) VGG-11

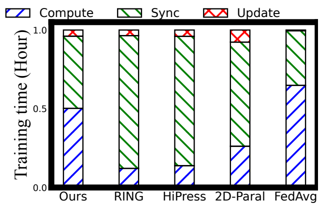
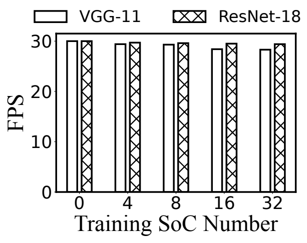

(b) ResNet-18

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


(a) Viking Village

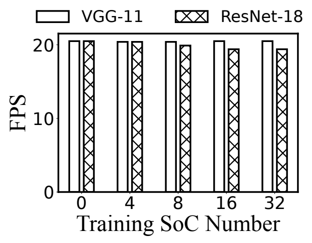

(b) Sewerage

Fig. 13. The coexistence of gaming performance (FPS) with training on different numbers of SoCs. "0" means no training workloads.

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 [76], 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 Coexistence performance

In this section, we conduct a thorough investigation into the coexistence of gaming performance (FPS) with training on different numbers of SoCs. We evaluate `SoCFlow+` with two open-sourced Unity games: Sewerage (2560×1440) [77] and Viking Village (1920×1080) [78], as illustrated in Figure 13. **The training workloads of `SoCFlow+` have a negligible impact on gaming performance.**

The coexistence of gaming performance with VGG11 and ResNet18 training workloads results in only a 1.1 FPS reduction for the Sewerage game, while the Vikingvillage game experiences a reduction of 0.6–1.7 FPS. This minimal degradation can be attributed to `SoCFlow+` utilizing only the idle mobile SoCs for training, thereby avoiding competition for computational resources with gaming SoCs.

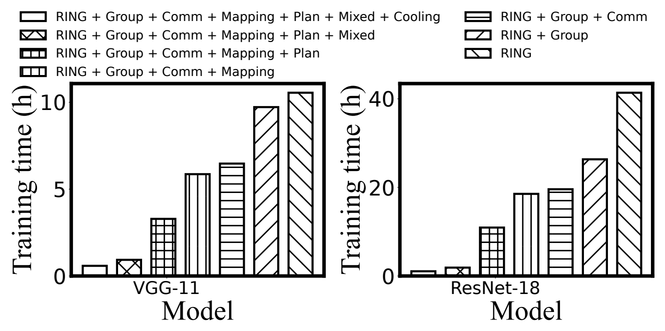


Fig. 14. Ablation study of SoCFlow+.

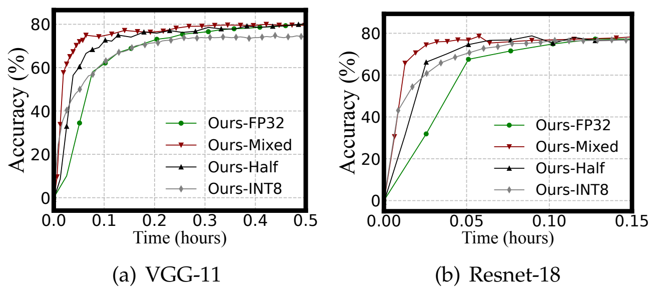


Fig. 15. Ablation study of the mixed-precision data-parallel training algorithm.

4.7 Ablation Study

Overall techniques. Figure 14 demonstrates the benefits of each individual technique in SoCFlow+. The rightmost bar is the same as baseline RING, while the leftmost one is SoCFlow+. The four steps in §3.1, the data-parallel mixed-precision training algorithm in §3.2, and the passive training-cooling co-design in §3.3 are represented by Group, Comm Mapping, Plan, Mixed, and Cool correspondingly. First, the group-wise parallelism with delayed aggregation can mitigate the network contention, achieving a 3.2–3.8× training speedup. Besides, the data-parallel mixed-precision training algorithm reduces training time by 3.53–5.78× because it can exploit both the CPU and NPU to train a model in parallel. Last, the passive training-cooling co-design achieves a 1.59–1.77× training speedup since it can mitigate the performance degradation caused by processor underclocking with minimal energy overhead.

Mixed-precision data-parallel training algorithm. Figure 15 further shows how SoCFlow+'s mixed-precision data-parallel training algorithm can improve convergence speed and accuracy. **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

SoCFlow+ with more advanced SoCs. The current prototype of SoCFlow+ is based on the Snapdragon 865. 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 [79], [80], [76]. 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 [76]. 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 on SoC-Cluster. To be noted, even with more advanced SoCs released, the design of SoCFlow+ will not become obsolete since the legacy SoCs still need can be harvested to reduce the manufacturing of new SoCs.

Applying SoCFlow+ to larger models. SoCFlow+ is motivated to leverage the idle resources of SoC-Cluster for edge-based model training and adaptation. Thereby, SoCFlow+ is mainly evaluated on small to medium-sized DNNs. However, with growing training demand of larger models like language models, SoC-Cluster and SoCFlow+ could be good complementary to datacenter-scale LLM training [81], [82], [83], [84]. While pre-training LLMs on SoC-Cluster is not yet feasible, we expect it to be affordable for downstream fine-tuning [85], [86]. In future work, we plan to explore the applicability of SoCFlow+ to large language models, which will involve further optimizing the communication and computation strategies within SoCFlow+ to handle the increased demands of these models.

6 Related Work

On-device training. Recently, there has been a trend to train a DNN model on mobile devices locally [7], [49], [10], [11], [12], [13], [87]. DeepType [13] proposes *incremental training* to effectively train a personalized deep learning model from a global model. Melon [11] 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 [26], [12]. SoCFlow+ is orthogonal to and compatible with those system-level optimizations.

Mixed-precision DNN training has been proposed to reduce training cost [88], [89], [90]. These approaches use lower-precision formats, such as INT8 and INT16, to represent the weights and activation generated during training. UI8 [63] 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 [60], [61], [36], [62], model parallelism [91], hybrid parallelism [92], and pipeline parallelism [93], [94], [23], [95], [96]. Besides, many solutions have been proposed to optimize communication efficiency between different workers [97], [98], [99], [100]. `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 [101], [102], [103], [104]. This approach can reduce significantly synchronization waiting time while still providing correctness guarantees.

Federated learning is also an emerging machine-learning paradigm [49], [9], [10], [37] 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 [35] and LAN-WAN aggregation [105] to save network traffic. Those studies inspire `SoCFlow+` group-wise parallelism with delayed aggregation. One key difference is that FL often handles non-IID data distribution, while `SoCFlow+` is designed for common distributed training scenarios where the developers tend to partition the data across SoCs in IID manner.

Junkyard Computing. A few recent works [106], [107] 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. Most of the `SoCFlow+`'s techniques are directly applicable to junkyard computing, though unique challenges need to be addressed, such as the low reliability of obsolete smartphones, even scarcer network, and thermal control.

7 CONCLUSION

In this paper, we introduced `SoCFlow+`, a pioneering framework designed to facilitate the efficient training of deep learning models on SoC-Cluster servers. `SoCFlow+` overcomes the challenges posed by restricted cross-SoC network bandwidth through the integration of two innovative strategies: group-wise parallelism with delayed aggregation and a data-parallel mixed-precision training algorithm. These methodologies enable `SoCFlow+` to deliver training performance that is both efficient and scalable. Our comprehensive experimental evaluation of `SoCFlow+` reveals its superior performance over existing methodologies in terms of training velocity while concurrently maintaining model accuracy.

ACKNOWLEDGMENTS

This work was supported by National Key R&D Program of China (No.2022ZD0119103) and NSFC (No.62325201 and No.62102045). Mengwei Xu and Gang Huang are the corresponding authors.

REFERENCES

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

[2] L. Zhang, Z. Fu, B. Shi, X. Li, R. Lai, C. Chen, A. Zhou, X. Ma, S. Wang, and M. Xu, "More is different: Prototyping and analyzing a new form of edge server with massive mobile socs," *USENIX ATC*, 2024.

[3] L. Zhang, S. Wang, and M. Xu, "High-density mobile cloud gaming on edge soc farms," *USENIX ATC*, 2024.

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

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

[6] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu, "From cloud to edge: a first look at public edge platforms," in *Proceedings of the 21st ACM Internet Measurement Conference*, 2021, pp. 37–53.

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

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan *et al.*, "Towards federated learning at scale: System design," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 374–388, 2019.

[9] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[10] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[11] Q. Wang, M. Xu, C. Jin, X. Dong, J. Yuan, X. Jin, G. Huang, Y. Liu, and X. Liu, "Melon: Breaking the memory wall for resource-efficient on-device machine learning," 2022.

[12] D. Xu, M. Xu, Q. Wang, S. Wang, Y. Ma, K. Huang, G. Huang, X. Jin, and X. Liu, "Mandheling: mixed-precision on-device dnn training with dsp offloading," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 214–227.

[13] M. Xu, F. Qian, Q. Mei, K. Huang, and X. 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*, vol. 2, no. 4, pp. 1–26, 2018.

[14] J. Han, Y. Ma, Q. Mei, and X. Liu, "Deeprec: On-device deep learning for privacy-preserving sequential recommendation in mobile commerce," in *Proceedings of the Web Conference 2021*, 2021, pp. 900–911.

[15] W. Li, Q. Xia, H. Cheng, K. Xue, and S.-T. Xia, "Vertical semi-federated learning for efficient online advertising," *arXiv preprint arXiv:2209.15635*, 2022.

[16] O. A. Wahab, G. Rjoub, J. Bentahar, and R. Cohen, "Federated against the cold: A trust-based federated learning approach to counter the cold start problem in recommendation systems," *Information Sciences*, vol. 601, pp. 189–206, 2022.

[17] P. Georgiev, N. D. Lane, K. K. Rachuri, and C. 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*, 2014, pp. 295–309.

[18] L. N. Huynh, Y. Lee, and R. K. 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*, 2017, pp. 82–95.

[19] F. Jia, D. Zhang, T. Cao, S. Jiang, Y. Liu, J. Ren, and Y. 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*. Association for Computing Machinery New York, NY, USA, 2022, pp. 209–221.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[21] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters," in *14th USENIX Symposium on Operating Systems Design and Implementation*, 2020, pp. 463–479.

[22] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[23] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[24] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[25] D. Cai, Q. Wang, Y. Liu, Y. Liu, S. Wang, and M. 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*, 2021, pp. 31–36.

[26] A. Das, Y. D. Kwon, J. Chauhan, and C. 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*. IEEE, 2022, pp. 533–538.

[27] Q. Zhang, X. Li, X. Che, X. Ma, A. Zhou, M. Xu, S. Wang, Y. Ma, and X. Liu, "A comprehensive benchmark of deep learning libraries on mobile devices," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 3298–3307.

[28] Q. Pei, S. Chen, Q. Zhang, X. Zhu, F. Liu, Z. Jia, Y. Wang, and Y. Yuan, "Cooledge: hotspot-relievable warm water cooling for energy-efficient edge datacenters," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 814–829.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[30] V. Chandrasekaran, S. Banerjee, D. Perino, and N. Kourtellis, "Hierarchical federated learning with privacy," *arXiv preprint arXiv:2206.05209*, 2022.

[31] N. Mhaisen, A. A. Abdellatif, A. Mohamed, A. Erbad, and M. Guizani, "Optimal user-edge assignment in hierarchical federated learning based on statistical properties and network topology constraints," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 1, pp. 55–66, 2021.

[32] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, T. Yu *et al.*, "Mnn: A universal and efficient inference engine," *arXiv preprint arXiv:2002.12418*, 2020.

[33] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *12th USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 265–283.

[34] Y. Bai, C. Li, Q. Zhou, J. Yi, P. Gong, F. Yan, R. Chen, and Y. Xu, "Gradient compression supercharged high-performance data parallel dnn training," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 359–375.

[35] K. Jayaram, V. Muthusamy, G. Thomas, A. Verma, and M. Purcell, "Adaptive aggregation for federated learning," *arXiv preprint arXiv:2203.12163*, 2022.

[36] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[37] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[38] J. Song, J. Yim, J. Jung, H. Jang, H.-J. Kim, Y. Kim, and J. 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*, 2023, pp. 560–573.

[39] M. Xu, L. Zhang, and S. Wang, "Position paper: Renovating edge servers with arm socs," in *2022 IEEE/ACM 7th Symposium on Edge Computing*. IEEE Computer Society, 2022, pp. 216–223.

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

[41] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "Antman: Dynamic scaling on gpu clusters for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation*, 2020, pp. 533–548.

[42] W. Zhang, B. Chen, Z. Han, Q. Chen, P. Cheng, F. Yang, R. Shu, Y. Yang, and M. Guo, "Pilotfish: Harvesting free cycles of cloud gaming with deep learning training," in *2022 USENIX Annual Technical Conference*, 2022, pp. 217–232.

[43] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[44] M. Wang, S. Rasoulinezhad, P. H. Leong, and H. K.-H. So, "Niti: Training integer neural networks using integer-only arithmetic," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 3249–3261, 2022.

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

[46] "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.

[47] N. Dryden, N. Maruyama, T. Moon, T. Benson, A. Yoo, M. Snir, and B. Van Essen, "Aluminum: An asynchronous, gpu-aware communication library optimized for large-scale training of deep neural networks on hpc systems," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2018.

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

[49] D. Cai, Y. Wu, S. Wang, F. X. Lin, and M. Xu, "Autofednlp: An efficient fednlp framework," *arXiv preprint arXiv:2205.10162*, 2022.

[50] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3043–3052.

[51] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Advances in neural information processing systems*, vol. 24, 2011.

[52] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[53] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[54] Y. You, Y. Wang, H. Zhang, Z. Zhang, J. Demmel, and C.-J. Hsieh, "The limit of the batch size," *arXiv preprint arXiv:2006.08517*, 2020.

[55] R. Ge, S. M. Kakade, R. Kidambi, and P. Netrapalli, "The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares," *Advances in neural information processing systems*, vol. 32, 2019.

[56] K. You, M. Long, J. Wang, and M. I. Jordan, "How does learning rate decay help modern neural networks?" *arXiv preprint arXiv:1908.01878*, 2019.

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

[58] P. M. Pardalos, T. Mavridou, and J. Xue, "The graph coloring problem: A bibliographic survey," in *Handbook of combinatorial optimization*. Springer, 1998, pp. 1077–1141.

[59] A. Bar-Noy and G. Kortsarz, "Minimum color sum of bipartite graphs," *Journal of Algorithms*, vol. 28, no. 2, pp. 339–365, 1998.

[60] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[61] M. Li, "Scaling distributed machine learning with system and algorithm co-design," Ph.D. dissertation, PhD thesis, Intel, 2017.

[62] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[63] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1969–1979.

[64] Y. Zhang, C. Li, and M. Pan, "Design and performance research of integrated indirect liquid cooling system for rack server," *International Journal of Thermal Sciences*, vol. 184, p. 107951, 2023.

[65] B. Watson and V. K. Venkiteswaran, "Universal cooling of data centres: a cfd analysis," *Energy Procedia*, vol. 142, pp. 2711–2720, 2017.

[66] M. K. Mehmet-Ali, J. F. Hayes, and A. K. Elhakeem, "Traffic analysis of a local area network with a star topology," *IEEE Transactions on Communications*, vol. 36, no. 6, pp. 703–712, 1988.

[67] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[68] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[69] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[70] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision*, December 2015.

[71] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[72] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "Cinic-10 is not imagenet or cifar-10," *arXiv preprint arXiv:1810.03505*, 2018.

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

[74] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep Gradient Compression: Reducing the communication bandwidth for distributed training," in *The International Conference on Learning Representations*, 2018.

[75] A. Olmo, S. Sreedharan, and S. Kambhampati, "Gpt3-to-plan: Extracting plans from text using gpt-3," *arXiv preprint arXiv:2106.07131*, 2021.

[76] "Ai benchmark." https://ai-benchmark.com/ranking.html, 2023.

[77] "Sewer/underground — unity asset store." https://assetstore.unity.com/packages/3d/environments/sewer-underground-modularpack-v4-0-112692, 2022.

[78] "Viking village urp — unity asset store." https://assetstore.unity.com/packages/essentials/tutorial-projects/viking-village-urp-29140, 2022.

[79] "Snapdragon 8 gen 1." https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-1-mobile-platform, 2023.

[80] "Snapdragon 8 gen 2." https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-2-mobile-platform, 2023.

[81] M. Xu, W. Yin, D. Cai, R. Yi, D. Xu, Q. Wang, B. Wu, Y. Zhao, C. Yang, S. Wang *et al.*, "A survey of resource-efficient llm and multimodal foundation models," *arXiv preprint arXiv:2401.08092*, 2024.

[82] R. Yi, L. Guo, S. Wei, A. Zhou, S. Wang, and M. Xu, "Edgemoe: Fast on-device inference of moe-based large language models," *arXiv preprint arXiv:2308.14352*, 2023.

[83] P. Zeng, Z. Ning, J. Zhao, W. Cui, M. Xu, L. Guo, X. Chen, and Y. Shan, "The cap principle for llm serving," *arXiv preprint arXiv:2405.11299*, 2024.

[84] J. Yuan, C. Yang, D. Cai, S. Wang, X. Yuan, Z. Zhang, X. Li, D. Zhang, H. Mei, X. Jia *et al.*, "Mobile foundation model as firmware," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024, pp. 279–295.

[85] D. Cai, Y. Wu, S. Wang, F. X. Lin, and M. Xu, "Efficient federated learning for modern nlp," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–16.

[86] M. Xu, Y. Wu, D. Cai, X. Li, and S. Wang, "Fwdllm: Efficient federated finetuning of large language models with perturbed inferences," *USENIX ATC*, 2024.

[87] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, "On-edge multi-task transfer learning: Model and practice with data-driven task allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1357–1371, 2020.

[88] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[89] X. Chen, X. Hu, H. Zhou, and N. Xu, "Fxpnet: Training a deep convolutional neural network in fixed-point representation," in *2017 International Joint Conference on Neural Networks*. IEEE, 2017, pp. 2494–2501.

[90] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[91] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[92] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks." *Proceedings of Machine Learning and Systems*, vol. 1, pp. 1–13, 2019.

[93] S. Athlur, N. Saran, M. Sivathanu, R. Ramjee, and N. Kwatra, "Varuna: scalable, low-cost training of massive deep learning models," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 472–487.

[94] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[95] F. Xu, Y. Qin, L. Chen, Z. Zhou, and F. Liu, "Dnn: Achieving predictable distributed dnn training with serverless architectures," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 450–463, 2022.

[96] H. Zheng, F. Xu, L. Chen, Z. Zhou, and F. Liu, "Cynthia: Cost-efficient cloud resource provisioning for predictable distributed deep neural network training," in *Proceedings of the 48th International Conference on Parallel Processing*, ser. ICPP '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3337821.3337873

[97] Y. Chen, Y. Peng, Y. Bao, C. Wu, Y. Zhu, and C. Guo, "Elastic parameter server load distribution in deep learning clusters," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 507–521.

[98] A. Elgabli, J. Park, A. S. Bedi, M. Bennis, and V. Aggarwal, "Gadmm: Fast and communication efficient framework for distributed machine learning." *J. Mach. Learn. Res.*, vol. 21, no. 76, pp. 1–39, 2020.

[99] S. H. Hashemi, S. Abdu Jyothi, and R. Campbell, "Tictac: Accelerating distributed deep learning with communication scheduling," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 418–430, 2019.

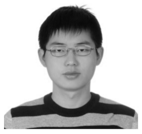
[100] A. Jangda, J. Huang, G. Liu, A. H. N. Sabet, S. Maleki, Y. Miao, M. Musuvathi, T. Mytkowicz, and O. Saarikivi, "Breaking the computation and communication abstraction barrier in distributed machine learning workloads," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 402–416.

[101] X. Zhao, A. An, J. Liu, and B. X. Chen, "Dynamic stale synchronous parallel distributed training for deep learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1507–1517.

[102] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," *Advances in neural information processing systems*, vol. 26, 2013.

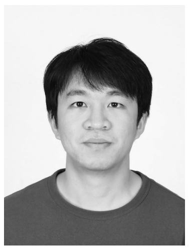
[103] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[104] Y. Zhou, Y. Yu, W. Dai, Y. Liang, and E. Xing, "On convergence of model parallel proximal gradient algorithm for stale synchronous parallel system," in *Artificial Intelligence and Statistics*. PMLR, 2016, pp. 713–722.
[105] J. Yuan, M. Xu, X. Ma, A. Zhou, X. Liu, and S. Wang, "Hierarchical federated learning through lan-wan orchestration," *arXiv preprint arXiv:2010.11612*, 2020.
[106] B. Na, J. Jang, S. Park, S. Kim, J. Kim, M. S. Jeong, K. C. Kim, S. Heo, Y. Kim, and S. Yoon, "Scalable smartphone cluster for deep learning," *arXiv preprint arXiv:2110.12172*, 2021.
[107] J. Switzer, G. Marcano, R. Kastner, and P. 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*, 2023, pp. 400–412.

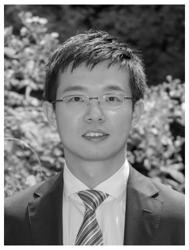

Mengwei Xu is an associate professor in the computer science department at Beijing University of Posts and Telecommunications. His research interests cover the broad areas of mobile computing, edge computing, artificial intelligence, and system software.

Daliang Xu is now a Ph.D. student in the School of Computer Science, Peking University, Beijing, China. His research interests lie in mobile computing and software engineering. Web page: https://daliangxu.github.io/

Chiheng Lou is currently a Ph.D. student in the School of Computer Science, Peking University, Beijing, China. His research interests include large language models and serverless computing.


Li Zhang is a Ph.D. student in the School of Computer Science, Beijing University of Posts and Telecommunication, Beijing, China. His research interests lie in system software of edge computing and mobile computing.

Gang Huang is a full professor in the School of Computer Science, Peking University. His research interests are in the area of middleware of cloud computing and mobile computing. He is a member of IEEE.


Xin Jin (Senior Member, IEEE) received the PhD degree from Princeton University in 2016. He is currently an Associate Professor (with Tenure) in School of Computer Science, Peking University. His research interests include computer systems, networking, and cloud computing. He received USENIX FAST Best Paper Award (2019) and USENIX NSDI Best Paper Award (2018).

Xuanzhe Liu is a Full Professor in the School of Computer Science at Peking University, Beijing, China. His research interests mainly fall in service-based software engineering and systems. Most of his recent efforts have been published at prestigious conferences including WWW, ICSE, FSE, SOSP, SIGCOMM, NSDI, MobiCom, MobiSys, and in journals including ACM TOSEM/TOIS and IEEE TSE/TMC/TSC. He is a distinguished member of the ACM and the CCF, and a senior member of the IEEE. Web page: http://www.liuxuanzhe.com/.