MTrain: Enable Efficient CNN Training on Heterogeneous FPGA-based Edge Servers

Yue Tang, Alex K. Jones, *Fellow, IEEE*, Jinjun Xiong, *Fellow, IEEE*, Peipei Zhou, *Senior Member, IEEE*, and Jingtong Hu, *Senior Member, IEEE*

***Abstract*—FPGA-based edge servers are used in many applications in smart cities, hospitals, retail, etc. Equipped with heterogeneous FPGA-based accelerator cards, the servers can be implemented with multiple tasks including efficient video prepossessing, machine learning algorithm acceleration, etc. These servers are required to implement inference during the daytime while re-training the model during the night to adapt to new environments, domains, or new users. During the re-training, conventionally, the incoming data are transmitted to the cloud, and then the updated machine learning models will be transferred back to the edge server. Such a process is inefficient and cannot protect users' privacy, so it is desirable for the models to be directly trained on the edge servers. Deploying convolutional neural network (CNN) training on heterogeneous resource-constrained FPGAs is challenging since it needs to consider both the complex data dependency of the training process and the communication bottleneck among different FPGAs. Previous multi-accelerator training algorithms select optimal scheduling strategies for data parallelism, tensor parallelism, and pipeline parallelism. However, pipeline parallelism cannot deal with batch normalization (BN) which is an essential CNN operator, while purely applying data parallelism and tensor parallelism suffers from resource under-utilization and intensive communication costs. In this work, we propose MTrain, a novel multi-accelerator training scheduling strategy that transfers the training process into a multi-branch workflow, thus independent sub-operations of different branches are executed on different training accelerators in parallelism for better utilization and reduced communication overhead. Experimental results show that we can achieve efficient CNN training on heterogeneous FPGA-based edge servers with 1.07x-2.21x speedup under 15 GB/s peer-to-peer bandwidth compared to the state-of-the-art work.**

***Index Terms*—edge server, heterogeneous FPGAs, CNN training.**

I. Introduction

FPGA-BASED edge servers are widely used in many applications in smart cities [1], [2], hospitals [3], retail [4], robotics [5], etc. Composed of heterogeneous accelerators, the servers can be implemented with multiple artificial intelligence (AI) tasks. For example, the video machine-learning streaming server (VMSS) [3], an edge server equipped with Xilinx Alveo U50LV+U30 FPGAs is proposed to build efficient video analytics in smart cities. The U30 FPGA card is designed for a series of video pre-processing tasks like video decoding, frame buffering, image crop, and scaling, while the U50LV FPGA is loaded with highly optimized machine learning engines and serves machine learning plugins for accelerated inference. The server is required to achieve exchangeability, i.e. implementing inference during the daytime and re-training the CNN model during the night so that the model can be adapted to new environments, domains, or new users. Conventionally, when the environments, tasks, or users change, data needs to be collected from the edge FPGAs and transmitted to the cloud. The re-training is implemented in the cloud, and the updated model is transmitted back from the cloud to the edge. The whole process is inefficient and cannot protect users' privacy [6], [7]. Therefore, it is desirable to scale the edge servers from inference into CNN training tasks so that they can continuously and locally learn from new data. Currently, many algorithms [7]–[12] have been proposed to achieve CNN training on local devices with high accuracy and low edge-to-cloud communication overhead. However, how to accelerate the training algorithms on the hardware side given the resource constraints of the target FPGA edge servers has not been well-investigated.

Recently, several designs [13]–[15] have been proposed to achieve efficient CNN training directly on a single FPGA. Those accelerators are well-developed, i.e. each training operation can achieve high training throughput on a single device. Accurate resource and performance models are also established to explore design parameters and estimate execution latency for each operation. To enable efficient CNN training on heterogeneous FPGA-based edge servers, it is necessary to develop an effective scheduling strategy to map the training process on these well-developed accelerators. However, it is challenging since it needs to consider both the computation complexity of the training process and the communication bottleneck among different accelerators. First, unlike inference which only involves forward propagation (FP), the training process includes FP, backward propagation (BP), and weight update (WU) with more types of operations and more complex data dependencies [15]. Fig. 1 (a) shows the data dependency within the training process of a CNN composed of a convolutional (Conv) layer, a batch normalization (BN) layer, and a fully connected (FC) layer. The circles represent operations, while the boxes represent tensors including activation (e.g. A_1) in FP (red lines), loss (e.g. L_2) in BP (black lines), weights

This work is supported in part by NSF awards #2122320, #2133267, #2213701, #2217003, #2235364, #2324864, #2324937, #2328972, #2329704, #2229873, NIH R01EB033387.

Yue Tang, and Jingtong Hu are with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261, USA (e-mail: yut51@pitt.edu; jthu@pitt.edu).

Jinjun Xiong is with the Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY 14260, USA (e-mail: jinjun@buffalo.edu).

Alex Jones is with the Department of Electrical Engineering and Computer Science Department, Syracuse University, 4-206 Center for Science and Technology, Syracuse, NY 13244, USA (e-mail: akj@syr.edu).

Peipei Zhou is with the School of Engineering, Brown University, 345 Brook Street, Providence, RI 02912, USA (e-mail: peipei_zhou@brown.edu).

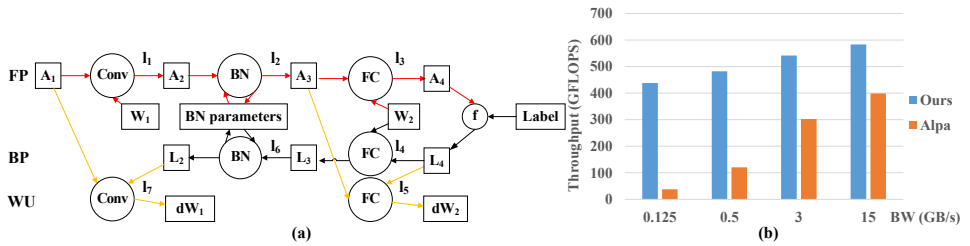

Fig. 1. CNN training involves complex data dependency. (a) The CNN training process. Circles represent operations, and boxes represent tensors. Arrows represent data dependency. FP is in red, BP is in black, and WU is in yellow. (b) Comparison of the DP/TP sharding algorithm in Alpa [23] and our design training the TinyYoloV3 network on the VMSS edge server. Both of them are tested on the same FPGA backend. It shows that the DP/TP-based mapping algorithm leads to sub-optimal training performance compared to our work on resource-limited edge servers.

gradient (e.g. dW_1) in WU (yellow lines) and weights (e.g. W_1). Activation and loss are immediate features in the training process. The activation in FP and loss in BP need to be used to calculate weights gradient in WU. Unlike cloud servers with abundant resources to handle such computation complexity, edge servers have restricted computation and memory resources. Edge server inference can apply quantized data to reduce the computation overhead on the limited resources without accuracy reduction [16], [17], while training with floating-point is preferred in most realistic applications to guarantee the model accuracy [15], [18]. Therefore, given the high complexity of the training process, finding the best distribution of computations on multiple resource-limited accelerators is non-trivial even though every single accelerator can achieve high throughput and low latency [19]. Second, to achieve high throughput and balanced execution, the training process needs to be partitioned and distributed on different FPGAs, so data needs to be scattered and gathered among FPGAs. With FP, BP, and WU, CNN training involves higher communication intensity [19]. Compared to high-performance cloud environments, edge servers always suffer from lower peer-to-peer (P2P) communication bandwidth. For example, the NVLink used in the Amazon EC2 cloud [20] achieves up to 300 GB/s GPU P2P bandwidth [21]. In edge servers, PCIe is widely equipped for P2P interconnection [3], [4]. We have tested the PCIe bandwidth on the UIUC XACC testbed [22], and the P2P bandwidth is only 3 GB/s. The higher data transfer overhead of the training process and the lower P2P bandwidth of the edge servers lead to the non-trivial communication bottleneck.

To address these two challenges and achieve efficient CNN training, many scheduling algorithms [19], [23]–[27] have been proposed to automatically partition the training process and allocate different components to different accelerators so these accelerators can execute in parallel. Existing parallelization techniques are typically categorized as data, tensor, and pipeline parallelism which can be seen in Fig. 2 [28]. Data parallelism (DP) means that the training data is partitioned in the batch dimension across distributed accelerators. The model's weights are replicated and scattered to each accelerator. In tensor parallelism (TP), the computation of a Conv operator is partitioned alone by non-batch axes, and weights are distributed across accelerators. Immediate results for each layer need to be gathered and added up from all accelerators. Pipeline parallelism (PP) divides a batch into micro-batches and places different groups of layers onto individual accelerators. It pipelines the computation of these micro-batches. However, PP cannot support CNNs with batch-wise operations [25]. Purely adopting DP and TP leads to sub-optimal performance due to accelerator under-utilization and communication bottlenecks. Fig. 1 (b) shows the throughput comparisons of training the TinyYoloV3 network on the VMSS (U50LV+U30) edge server scheduled by Alpa [23] with DP/TP and by our works. The communication bandwidth (BW) between U50LV and U30 ranges from 0.125 GB/s to 15 GB/s. The DP and TP techniques achieve sub-optimal training throughput especially when the server suffers from low bandwidth. The detailed analysis will be illustrated in Sec. III-B.

To achieve efficient CNN training on resource-limited edge servers, we propose a novel automatic multi-accelerator CNN training scheduling strategy MTrain. We first transfer the CNN training process which involves complex data dependency into a multi-branch workflow, where computation-intensive operations are divided into independent sub-operations. Sub-operations without data dependency can be executed on different accelerators in parallel, and such a parallelism technique is called sub-operation parallelism. Second, we function the training scheduling problem into a multi-branch workflow to a multi-accelerator mapping problem. The motivation for this idea will be discussed in Sec. III-C.

Based on the proposed sub-operation parallelism, we only need to solve two problems: how to convert the training process into a multi-branch workflow with independent sub-operations, and how to allocate each sub-operation of the workflow on different training accelerators for better utilization. These two problems need to be jointly co-optimized. To address these problems, we propose a novel automatic multi-accelerator CNN training scheduling strategy MTrain. Our main contributions are as follows.

- We propose MTrain-Converting, a two-stage CNN training to multi-branch workflow converting approach to tackle the first problem. It can support BN operations for commonly used modern CNNs (Feature ④). The converting approach will be introduced in Sec. IV.
- We propose MTrain-Mapping, a multi-branch to multi-accelerator scheduling algorithm (Feature ①) to address the second problem. The algorithm solves the cross-

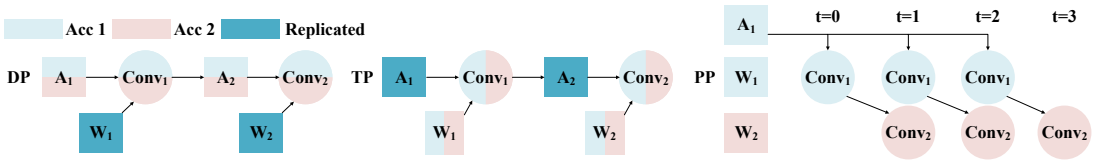

Fig. 2. Common parallelization techniques for training a CNN with two Conv layers. Only the FP pass is shown.

branch data dependency (Feature ②) and communication bottleneck considering the off-chip memory budget during FP/BP/WU (Feature ③). As shown in Fig 1 (b), different layers are executed in an asynchronous manner (Feature ⑤), and the scheduling algorithm is shown in Sec. V

- We achieve efficient CNN training on heterogeneous FPGA-based resource constraint edge servers (Feature ⑥). Compared with the SOTA work, HAP [27], MTrain achieves 1.07x-2.21x speedup under 15 GB/s P2P bandwidth. Detailed experimental results are shown in Sec. VI.

II. Related Works

A. Training on Local Edge Devices

As explained in Sec. I, it is desirable for edge FPGAs to continuously and locally learn from new data. Such on-device learning can directly improve model accuracy and efficiently adapt to new environments without ruining users' privacy. Currently, several algorithms have been proposed to enable edge devices to achieve domain adaption locally. For example, federated learning-based on-device learning algorithms [8], [9], [12] are designed to train from local data that are not independently and identically distributed. Contrastive learning-based on-device learning designs [7], [10], [11] have been explored to improve CNN models with limited labeled data. To efficiently implement these complex and fantastic software-level algorithms on our target edge servers, accelerating the training process in the hardware-level design is indispensable.

B. FPGA-based CNN Training Accelerators

To implement CNN training on target resource-limited edge servers with high throughput without modifying the training algorithms, a powerful training accelerator is required. Nowadays, several FPGA-based training accelerators have been proposed to achieve high throughput and energy efficiency. For example, DarkFPGA, a batch-level parallelism-based training accelerator is designed and implemented on the Maxeler MAX5 platform [13]. Venkataramanaiah et. al. present an FPGA accelerator for CNN training that uses high bandwidth memory (HBM) for efficient off-chip communication and supports various training operations such as residual connections, and stride-2 convolutions for modern CNNs [14]. EF-train, an efficient FPGA-based on-device CNN training accelerator has been proposed to achieve end-to-end training on resource-limited edge devices and supports operations like Conv, FC, and BN that are commonly used in modern CNNs [15]. These works have established accurate resource models to find optimal design parameters for a given FPGA and performance models to estimate the computation and intra-FPGA communication costs for a given operation.

C. Multi-accelerator Training Scheduling Algorithm

With well-developed single training accelerators and performance models to estimate the latency of arbitrary operations for a given FPGA, implementing CNN training on heterogeneous edge servers with high throughput also requires an efficient and effective scheduling algorithm. Nowadays, several scheduling algorithms have been proposed with the combination of DP and TP. Table I compares SOTA multi-accelerator training algorithms considering the 6 features mentioned in Sec. I. AccPar [19] proposes a layer-wise dynamic partition algorithm to find the optimal partition dimensions in TP when training on heterogeneous TPUs. FPDeep [24] and NADA [25] explore PP for training CNNs on homogeneous FPGA clusters. FPDeep pipelines in a fine-grained manner and stores immediate data on the FPGA chips to avoid dynamic random access memory DRAM access, and TP is involved to further balance the workload. Alpa [23] slices a homogeneous GPU cluster into a number of device meshes and uses a dynamic programming algorithm to assign training stages to meshes so that the stages can be implemented in a pipeline manner. Inside a mesh, an integer linear programming (ILP)-based sharding algorithm is developed to find the optimal DP/TP combination for different operations. DynaPipe [26] is the most recent PP-based algorithm that proposes a dynamic micro-batching approach to tackle CNN inputs with variant sequence lengths. HAP [27], the latest DP/TP-based work, jointly optimizes the sharding strategy, sharding ratios across heterogeneous GPUs, and the communication methods for tensor exchanges. An A*-based search algorithm is proposed to find the optimal DP/TP scheme, and the best sharding ratio among various devices is derived by formulating the problem as a linear programming problem.

Among them, FPDeep, NADA, and DynaPipe are based on PP. PP can reduce a large amount of Intra-layer communication costs [28]. However, PP cannot support CNNs with batch-wise operations like BN since it needs to calculate the means and variance of a batch of immediate features in FP and BP. The successive layers of BN cannot start until all micro-batches of BN are completed. Previous research has shown that BN is not only important in improving the performance of a CNN but is essential for being able to train the CNN [29]. Therefore, BN is indispensable in modern CNNs such as Vgg [30], ResNet [31], Yolo [32], etc. To support batch-wise operations, only non-PP techniques can be applied.

Among the rest works, Alpa can only adopt DP/TP, but as mentioned in Sec. I, the accelerator under-utilization and com-

TABLE I
COMPARISONS WITH SOTA MULTI-ACCELERATOR TRAINING SCHEDULING ALGORITHMS

Features	① Parallelism	② Memory	③ Cross-branch	④ BN	⑤ Asynchronous	⑥ Target Platform
AccPar [19] (2020)	TP	×	✓	✓	×	heterogeneous TPU cloud
FPDeep [24] (2020)	TP/PP	×	×	×	×	homogeneous FPGA cloud
NADA [25] (Unknown)	PP	✓	✓	×	×	homogeneous FPGA cloud
Alpa [23] (2022)	DP/TP/PP	✓	✓	✓	×	homogeneous GPU cloud
DynaPipe [26] (2024)	PP	✓	Unknown	×	✓	homogeneous GPU cloud
HAP [27] (2024)	DP/TP	×	✓	✓	×	heterogeneous GPU cloud
MTrain (Ours)	sub-operation parallelism	✓	✓	✓	✓	heterogeneous FPGA edge server

munication bottleneck lead to low training throughput. AccPar and HAP target resource-abundant TPU/GPU clusters, so the memory budget is not involved in their algorithms. Besides, the resource-abundant high-performance clouds applied in AccPar, Alpa, and HAP have superior P2P bandwidth. To achieve efficient CNN training on resource-constrained edge servers, a novel algorithm is necessary to solve the computation complexity and communication bottleneck. To the best of our knowledge, we are the first attempt to achieve efficient CNN training including both batch-wise and non-batch-wise operations on FPGA-based edge servers with limited resources and lower P2P bandwidth.

D. Mapping Multi-branch Networks with Cross-Branch Data Dependency on Multiple Accelerators

Currently, DNNs are rapidly evolving from streamlined single-modality single-task (SMST) to multi-modality multi-task (MMMT) with large variations for different layers and complex data dependencies among layers [33]. Such MMMT models contain multiple branches and involve complex inter-block connections between multiple backbones of different sizes. Several works [34], [35] have been proposed to efficiently map such multi-branch network inference on multiple accelerators. For example, M5 [35] explores flexible accelerator configurations and possible resource sharing among layers and maps MMMT models on homogeneous clusters in a pipelined manner. For the non-PP-based approach, H2H [34] proposes an iterative heuristic algorithm to map MMMT models on heterogeneous off-the-shelf FPGA-based accelerators with 4 steps including computation prioritized mapping under zero local DRAM assumption, weight locality optimization, activation transfer optimization, and data locality-aware re-mapping. The MMMT-Mapping algorithm enables independent layers to asynchronously run on different accelerators in parallel and achieves efficient model inference with cross-branch data dependency. Inspired by H2H, it is promising to apply the MMMT-mapping algorithm to allocate sub-operations generated by MTrain-Converting on heterogeneous accelerators, i.e. establish MTrain-Mapping based on MMMT-mapping. However, compared to MMMT inference, the multi-branch training workflow is more complicated. First, compared to inference, the data transfer includes activation in FP, loss in BP, and weight gradient in WU, which is more complicated. Second, the memory budget is more limited since training needs to store immediate activation and loss until the Conv layer passes WU. Therefore, we establish MTrain-Mapping on top of H2H and add further optimizations to address these two problems.

III. MOTIVATION

A. Definition of The CNN Training Scheduling Problem

We have a CNN model with a batch size of B, an edge cluster with N FPGAs, and M off-the-shelf training accelerator intellectual properties (IPs). The input includes the CNN model graph G_{model}, the FPGAs information $\{F_i\}$ $(i = 1, ..., N)$, and accelerator IP information $\{IP_j\}$ $(j = 1, ..., M)$. In the model graph $G_{model} = (V, E)$, the vertex V represents the operations of the CNN. Each operation node contains the following information: layer type (e.g. Conv, BN, etc.) of the operation, state (e.g. FP, BP, or WU), and parameters (e.g. number of input channels, number of output channels, feature map size, and weights kernel size of a Conv operation). Edge E shows data dependencies between different operations. The FPGA information contains the number of processing elements (PEs), on-chip block RAMs (BRAMs) size, off-chip memory (DRAM) size, on-chip to off-chip communication bandwidth of each FPGA, and the P2P bandwidth between different FPGAs. The inter-FPGA communication costs for two arbitrary FPGAs can be calculated by the output data size of an operation divided by the P2P bandwidth of these two FPGAs.

We adopt the well-developed FPGA accelerators [13], [15] as our preliminary works. These accelerators incorporate accurate resources and performance models that can be used to estimate the latency of a given operation. One FPGA is deployed with one accelerator, and it can select the design with the maximum single accelerator throughput. When the jth IP is deployed on the ith FPGA, given the resource constraints of F_i, i.e. DSPs and BRAMs numbers and the on-chip to off-chip communication bandwidth, the performance model and resource model of IP_j can automatically find the optimal design parameters and generate the latency, i.e. computation cost, executing each operation of a CNN. Our work aims to find the optimized scheduling strategy so that we can accelerate the CNN training process on a given FPGA-based edge server with high throughput.

It should be noted that given G_{model}, $\{F_i\}$, and $\{IP_j\}$, Mtrain provides a fixed scheduling scheme. For each training epoch in the on-device learning scenarios, such a scheme will be unchanged unless the architecture of the CNN model or the hardware configuration of the FPGA cluster changes. Therefore, this work provides a pre-computed optimal solution. This solution only needs to be generated once. Thus, the search time

of MTrain does not undermine the training performance and is not included in the overall throughput evaluation.

B. Limitations of DP/TP-based Training Scheduling algorithms

As mentioned in Sec. I, DP/TP suffers from accelerator under-utilization and communication bottlenecks. First, DP/TP always leads to idle stages in heterogeneous clusters and thus causes accelerator under-utilization. Fig. 3 (b) depicts how DP/TP schedules a 3-layer CNN shown in Fig. 3 (a) to 2 different accelerators: Acc1 and Acc2. In DP/TP, an operation is partitioned with a sharding ratio and then distributed to available accelerators based on the ratio. Circles 1.1 and 1.2 represent two sub-operations partitioned from layer 1 in Fig. 3 (a). 1.1 is mapped to Acc1, while 1.2 is mapped to Acc2. The box G represents that features from 1.1 and 1.2 are gathered to the host and scattered to 2.1 and 2.2. The yellow boxes represent the latency of the computation stage Lat_{Comp}, while the red boxes represent the latency of the communication stage Lat_{Comm}. In Conv operations of DP/TP, after all the accelerators finish the computation stage of the sub-operations, output features of a layer are gathered to a host device and then scattered to the accelerators as input features for the next layer. The BN operation costs little computation time, but data needs to be re-scattered before 3.1 and 3.2 start. The communication happens after all accelerators finish the computation stage of the layer, which means different accelerators execute one layer synchronously in DP/TP. However, the computation time for each layer varies among heterogeneous systems, so the computation time of each layer is the maximum computation time among the accelerators [27]. Therefore, as shown in Fig. 3 (b), Acc1 remains idle after 1.1 is finished until Acc2 completes 1.2, which is under-utilized.

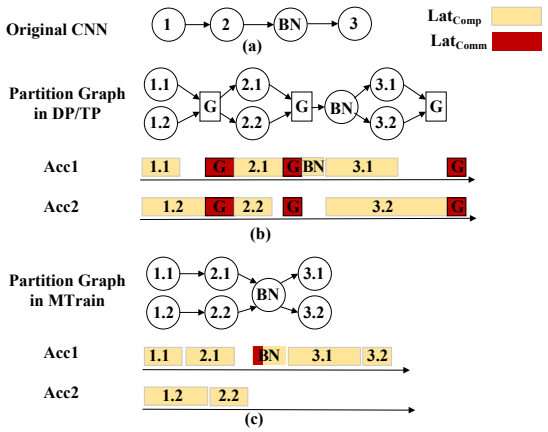

Fig. 3. Comparison between traditional DP/TP and our sub-operation parallelism-based technique. (a) A CNN with 3 Conv operations: layer 1, layer 2, and layer 3. It also has one BN operation. (b) DP/TP-based graph sharding and workload distribution. The index $a.b$ in the circle represents the bth sub-operation of the ath layer allocated on the bth accelerator. (c) Our work transfers the CNN into a multi-branch workflow with independent sub-operations on different branches and then proposes a multi-branch to multi-accelerator scheduling algorithm. $a.b$, the bth sub-operation of the ath layer, is free to be allocated on both Acc1 and Acc2. Compared to the DP/TP-based sharding algorithm, we achieve better accelerator utilization and less communication overhead.

Second, DP/TP suffers from intensive communication overhead. As shown in Fig. 3 (b), features from all accelerators are gathered and scattered after each layer. Such intensive communication costs lead to inefficient training performance. To enable efficient training on the edge servers, a novel scheduling strategy is necessary to address these two problems.

C. Sub-operation parallelism-based Training Scheduling algorithm

In this MTrain, we propose to transfer the CNN training process into a multi-branch workflow. The original computation-intensive operation (e.g. Conv and FC) is partitioned into sub-operations along the batch dimension. These sub-operations are independent and located on different branches. The arrows within a branch represent data dependencies of sub-operations. Sub-operations without data dependency can be executed on different accelerators in parallel, and such a parallelism technique is called sub-operation parallelism. Fig. 3 (c) shows an example of the mapping scheme of the CNN in Fig. 3 (a). The data flow in a mini-batch training is as follows. Each CNN layer is divided into batch-wise operation and non-batch-wise operation. For non-batch-wise operation, each sample inside a batch is calculated independently. For example, in Conv or FC, samples 1 and 2 conduct channel-level convolutional separately without any batch-level dependency. When a batch of data comes, a non-batch-wise operation can be partitioned into dependent sub-operations along batch-dimension with a partition ratio. These sub-operations are free to be scattered to different accelerators to execute in parallel or execute sequentially. For example, layer 1 is partitioned into sub-operations 1.1 and 1.2 that are deployed to Acc1 and Acc2, respectively. The input data for 1.1 is sent to Acc1, while the input data for 1.2 is sent to Acc2. Layer 3 is partitioned into sub-operations 3.1 and 3.2, and both of them are deployed to Acc1. Immediate features for layer 3 only pass Acc1. Batch-wise operation has data dependency over a mini-batch. For example, BN needs to calculate the means and variance of a batch of immediate features. Such operations will not be scattered to different accelerators, and one batch-wise operation is a sub-operation. For example, the BN in Fig. 3 (c) is deployed in Acc1, and data from its predecessors 2.1 and 2.2 are gathered in Acc1. In the proposed sub-operation parallelism, immediate data do not need to be gathered and scattered after each sub-operation, and only dependent sub-operations located on different accelerators require inter-device data communication. For example, in Fig. 3 (c), Lat_{Comm} only happens when the output data of 2.2 deployed to Acc2 are needed for BN located to Acc1. We consider the following data dependencies: immediate activation/loss between two adjacent sub-operations in FP/BP needs to be transferred, weights in FP are needed in BP, activation in FP and loss in BP are needed in WU, and weights gradients are accumulated to update weights. In the proposed sub-operation parallelism-based scheduling strategy, the latency for each sub-operation is the sum of Lat_{Comp} and Lat_{Comm}. With the latency for each sub-operation and the mapping scheme shown in Fig. 3 (c), we can calculate the overall latency and thus generate the training throughput.

Compared to DP/TP, the sub-operation parallelism proposed in MTrain has three features. First, sub-operations are executed asynchronously. As shown in Fig. 3 (c), sub-operation 2.1 can start after its predecessor 1.1 is completed without waiting for 1.2. Second, MTrain allows a larger design space such that two independent sub-operations are free to be allocated on both different accelerators and the same accelerator. The CNN training scheduling problem is transferred to find the optimal sub-operation to the heterogeneous accelerator mapping scheme. In an optimal mapping scheme, e.g. Fig. 3 (c), Acc1 is allowed to be implemented with sub-operations on different branches, i.e. 3.1 and 3.2 instead of strictly balancing 3.1 and 3.2 on Acc1 and Acc2 respectively. The asynchronous sub-operation execution and the larger design space lead to better utilization compared to DP/TP.

Third, MTrain has lower communication costs than DP/TP. The latency for one sub-operation includes the computation latency and the communication latency between the current sub-operation and its predecessor. We only consider data transfer between dependent sub-operations located on different accelerators without gathering features from all accelerators, so the communication overhead is reduced.

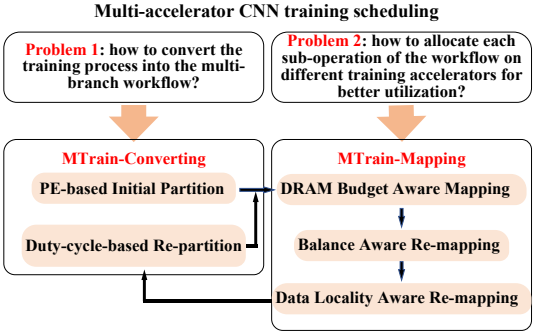

Fig. 4. The overview of MTrain. It includes MTrain-Converting to transfer the CNN training into a multi-branch workflow with independent sub-operations on different branches and MTrain-Mapping to allocate sub-operations on accelerators. MTrain-Converting includes 2 steps, and MTrain-Mapping includes 3 steps. The 5 steps form a close loop optimization workflow and work iteratively until no more beneficial scheduling scheme is acquired.

Based on such observation, to achieve efficient CNN training on heterogeneous FPGA-based edge servers, we only need to solve how to convert the training process into the multi-branch workflow and how to allocate each sub-operation on different. To address these problems, we propose a novel automatic multi-accelerator CNN training scheduling strategy MTrain, and the overview of our work is shown in Fig. 4.

IV. MTrain-Converting

In this section, we will introduce how we convert the CNN training process into a multi-branch workflow. As can be seen in Fig. 1 (a), the 3-layer CNN has 7 operations, i.e. l_1-l_7, in FP/BP/WU. Assume FP is streamlined, while operations in BP and WU are located in different branches. If the number of independent operations is less than the number of accelerators, some accelerators will remain idle. Therefore, we partition operations into N sub-operations, where N is the number of accelerators. The computation-intensive operations such as Conv and FC are non-batch-wise. It means the output features are accumulated in channel, row, and column dimensions, but different samples of a batch are independent. Therefore, for then non-batch-wise operations, we divide them into N parts along the batch dimension with a partition ratio. Batch-wise operations like BN are not computation-intensive, so we do not partition and distribute them across accelerators. These operations function as cross points when N branches fuse. An example of a multi-branch workflow converted from the 3-layer CNN of Fig. 1 (a) is illustrated in Fig. 5. There are 12 operations, i.e. l_1-l_{12}, after conversion. A_1B_1 in l_1 and A_1B_2 in l_2 represent two independent tensors partitioned from A_1. We have analyzed all completed data dependencies during training which are as follows. In FP, a batch-wise operation like BN starts after data from all branches of its predecessors are gathered. For example, the outputs A_2B_1 of l_1 and A_2B_2 of l_2 are fused in l_3. In BP, the loss is propagated back until the first layer. Batch-wise operations (e.g. l_{10}) are also dependent on previous operations from all branches (e.g. l_4 and l_5). In WU, gradients are calculated via activations in FP and loss in BP. For example, l_{11} is dependent on A_1B_1 and L_2B_1. Besides, weights are updated after gradients of all branches are calculated and gathered. For example, l_{11} and l_{12} finish after dW_2B_1 and dW_2B_2 are gathered to accelerators where $l1$ and $l2$ are placed. The above-mentioned process can also be applied to CNNs with multiple branches such as ResNet.

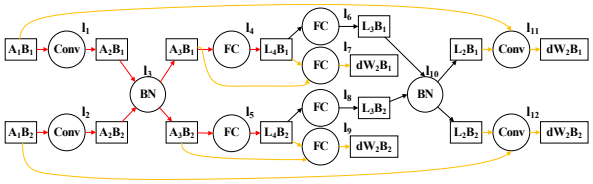

Fig. 5. An example of a multi-branch workflow transferred from the training process. The sub-operations without data dependency can be executed in parallel on different accelerators. It shows that the DP/TP-based mapping algorithm leads to sub-optimal training performance compared to our work on resource-limited edge servers.

An inappropriate partition ratio in the conversion will lead to under-utilization of the accelerators and thus lead to sub-optimal mapping results. However, finding an optimal partition ratio on a N-accelerator cluster when the batch size is B is an NP-hard problem and is time-consuming. Therefore, we propose a two-stage CNN training to a multi-branch workflow converting algorithm, MTrain-Converting, to iteratively search for a near-optimal solution. As shown in Fig. 4, it involves PE-based Initial Partition and Duty-cycle-based Re-partition.

A. Definition of The Model Converting Problem

As introduced in Sec. III-A, we have a CNN model with a batch size of B, an edge cluster with N FPGAs, and M off-the-shelf training accelerator IPs. As shown in Alg. 1, We choose the IP with the best overall throughput of the CNN on the ith FPGA, and the FPGA deployed with the selected IP becomes an accelerator Acc_i. Each accelerator incorporates a performance model to estimate Lat_{Comp} for each sub-operation. We select a $step\ size$ as a search parameter to find

Algorithm 1 MTrain-Converting

Require:
$G_{model} = (V, E)$, $\{F_i\}$, $\{IP_j\}$, $step\ size$;
Ensure:
$\{Acc_i\}$, $G^*_{model} = (V^*, E^*)$, $G^*_{sys} = \{G^*_{Acc_i}\}$, Lat;
1: PE-BASED INITIAL PARTITION ();
2: $\{Acc_i\}$ = Accelerator Selection (G_{model}, $\{F_i\}$, $\{IP_j\}$);
3: $\{ratio_i\} = B * \{PE_i\}/sum(\{PE_i\})$;
4: G^*_{model} = Partition ($\{ratio_i\}$, G_{model});
5: G^*_{sys}, Lat = MTrain-Mapping ($\{Acc_i\}$, G^*_{model});
6: DUTY-CYCLE-BASED RE-PARTITION()
7: **Repeat**
8: Estimate and sort duty cycle $\{Duty_i\}$ for each accelerator, and $ratio_i$ is rearranged based on the sorted order;
9: $w = 1$;
10: **for** r in $[1, N]$, and $r \neq w$ **do**
11: $ratio_r += step\ size$;
12: $ratio_w -= step\ size$;
13: $G1^*_{model}$ = Partition ($\{ratio_i\}$, G_{model});
14: $G1^*_{sys}$, $Lat1$ = MTrain-Mapping ($\{Acc_i\}$, $G1^*_{model}$);
15: **end for**
16: Find the lowest $Lat1$ with the r, $G1^*_{model}$ and $G1^*_{sys}$;
17: **if** $Lat1 < Lat$ **do**
18: Update G^*_{model}, G^*_{sys}, $\{ratio_i\}$ and Lat, and go step 8;
19: **else**
20: $w += 1$, and go step 10;
21: **Until** no more beneficial mapping result after re-partition

the partition ratio. Our goal is to search for an appropriate ratio and generate the outputs including deployed accelerators on FPGAs $\{Acc_i\}$, the multi-branch workflow graph G^*_{model}, and $\{Acc_i\}$ to G^*_{model} mapping scheme G^*_{sys} that lead to the maximum training throughput, i.e. the minimum overall latency Lat.

B. PE-based Initial Partition

In the first stage of Alg. 1, we first select accelerator IPs that have the maximum throughput and deploy them on the FPGAs (step 2). Then, we generate an initial ratio based on the number of PEs in each FPGA (step 3), convert G_{model} to G^*_{model} (step 4), and obtain the initial mapping results (step 5).

In step 2, for each FPGA F_i, we deploy each IP IP_j on it and test the CNN training throughput via the performance model of IP_j under the resource constraint of F_i. We do not consider DRAM costs in this step. The IP with the best throughput is deployed to F_i and becomes Acc_i. In step 3, the list of the ratio values $\{ratio_i\}$ is a series of non-negative integers that sum up to B, and the length of the list is the number of accelerators N. The maximum throughput is always achieved when the workload is well-balanced and the number of PEs, i.e. DSPs, can approximate the maximal computation performance of the accelerator. Therefore, we initialize the ratio for the list elements as the ratio for the number of DSPs for each accelerator. In step 4, the batch-wise operations in G_{model} are partitioned into N parts, where the ith part becomes an individual sub-operation with the batch size of $ratio_i$. Non-batch-wise operations remain unchanged. An example of the converted workflow graph $G^*_{model} = (V^*, E^*)$ is illustrated in Fig. 1 (b). In step 5, we apply the MTrain-Mapping algorithm in Sec V to map G^*_{model} on $\{Acc_i\}$. It generates the initial G^*_{sys} and the CNN training latency Lat.

C. Duty-cycle-based Re-partition

Although the PE-based partition balances the operation workload based on computation resources for each accelerator, the actual layer performance is not proportional to the number of PEs. Besides, the complex inter-layer communication makes G^*_{sys} unbalanced. Therefore, some accelerators remain idle under such a partition ratio. Fig. 6 (a) shows such a case. When the original ratio is $[4 : 2]$. Allocating l_2 to Acc_2 (the grey box) costs more time than allocating l_2 to Acc_1 after l_1. Therefore, the optimal mapping scheme searched by MTrain-Mapping is to allocate l_1-l_3 on Acc_1 which is shown in the pink boxes. If we reduce B_2 by a step size 1, Acc_2 is capable of completing l_2 before l_1 finishes. As can be seen in Fig. 6 (b), the workload is well-balanced after re-partitioning the ratio into $[5 : 1]$, and the overall latency is decreased.

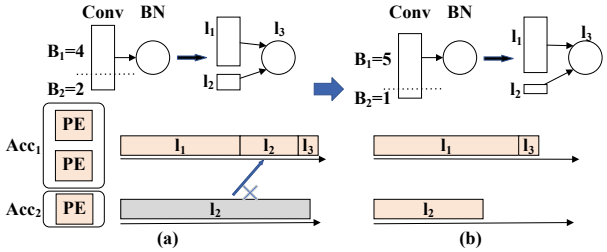

Fig. 6. An example of Duty-cycle-based Re-partition. Acc_1 contains double PEs compared to Acc_2 and has a better performance. (a) The workload is not well-balanced under the initial ratio $[4 : 2]$, and Acc_2 remains idle for the Conv and BN operations. (b) The workload is balanced after re-partition with a new ratio $[5 : 1]$, and the overall latency is reduced.

Therefore, in the second stage, we re-partition by reducing the ratio value that corresponds to the most idle accelerator, i.e. the accelerator with the least duty cycle. It should be noted that every time the partition ratio is changed, we conduct MTrain-Mapping for a new mapping scheme and duty cycle. The proposed Duty-cycle-based Re-partition function is shown in steps 6-21. In step 8, we estimate the duty cycle from G^*_{sys} and sort it which is shown as $\{Duty_i\}$. From steps 9-20, we first reduce the ratio value $ratio_w$ starting from the element that corresponds to the least duty cycle, i.e. $w = 1$. Second, for each of the rest elements r, we try to add $ratio_r$ with a step size and reduce the same value for $ratio_w$. Third, we re-map the converted graph $G1^*_{model}$ which is the workflow converted from the updated partition ratio, and select the partition ratio with the lowest latency $Lat1$. If the latency is reduced, we update the partition ratio, converted workflow, and the mapping scheme. If not, we will reduce the ratio value from the one corresponding to the second least-duty-cycle accelerator (step 20). The algorithm stops until no ratio value can be modified for better results.

V. MTRAIN-MAPPING

After a multi-branch workflow is generated with a given partition ratio, a mapping algorithm is needed to allocate each operation on different accelerators. As mentioned in Sec. II-D, MTrain-Mapping is inspired by the MMMT mapping algorithm H2H to address the problem but adds more optimizations to solve the non-trivial data transfer in FP/BP/WU and the DRAM budget problems. As illustrated in Fig. 4, MTrain-Mapping is composed of 3 procedures: DRAM Budget Aware Mapping, Balance Aware Re-mapping, and Data Locality Aware Re-mapping.

A. Definition of The Model Mapping Problem

Same with H2H, the converted model has complicated dependencies, especially for cross-talk connections. As shown in Alg. 2, the inputs for our algorithm are G^*_{model} and $\{Acc_i\}$, where the outputs are G^*_{sys} and Lat. In the mapping scheme $G^*_{sys} = \{G^*_{Acc_i}\}$, each sub-graph $G^*_{Acc_i}$ is a computation graph representing the layers' execution scheduling on the ith FPGA accelerator Acc_i. Each sub-graph is empty without any mapping at the beginning. After mapping, $G^*_{Acc_i}$ contains nodes located in Acc_i from G^*_{model} in their execution order. An example of G^*_{sys} after mapping is shown in Fig. 3 (c).

Algorithm 2 MTrain-Mapping

Require:
G^*_{model}, $\{Acc_i\}$;
Ensure:
$G^*_{sys} = \{G^*_{Acc_i}\}$, Lat;
1: DRAM Budget Aware Mapping()
2: **for** n in G^*_{model}, and n does not have predecessors **do**
3: Enumerate all possible mappings for $\{Acc_i\}$;
4: Check the local DRAM budget for each mapping candidate;
5: Calculate ΔLat;
6: Choose the mapping with minimum ΔLat;
7: **end for**
8: Update Lat, DRAM budget, and G^*_{sys};
9: Remove the mapped nodes in G^*_{model};
10: Balance Aware Re-mapping()
11: Generate the well-balanced mapping scheme G^B_{sys};
12: **Repeat**
13: **for** n in G^*_{model}, and n is mapped on different accelerators in G^*_{sys} and G^B_{sys} **do**
14: Attempt to re-map n to the accelerator on G^B_{sys};
15: Calculate Lat after re-mapping;
16: Accept the re-mapping if its latency is less than Lat;
17: Update G^*_{sys} and Lat;
18: **end for**
19: **Until** no more beneficial re-mapping sub-operations
20: Data Locality Aware Re-mapping()
21: **Repeat**
22: **for** n in G^*_{model} **do**
23: Attempt to re-map n to its predecessor's or successor's accelerator;
24: Calculate Lat after re-mapping;
25: Accept the re-mapping if its latency is less than Lat;
26: Update G^*_{sys} and Lat;
27: **end for**
28: **Until** no more beneficial re-mapping sub-operations

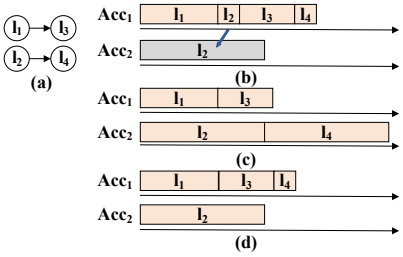

Fig. 7. An example of Balance Aware Re-mapping. (a) An example of G^*_{model} with 4 sub-operations. l_1 and l_3 have data dependency, and l_2 and l_4 have data dependency. l_1 and l_2 are partitioned from the same CNN layer, while l_3 and l_4 are from the same CNN layer. (b) The mapping scheme after DRAM Budget Aware Mapping. Acc_1 has better performance than Acc_2. All sub-operations are mapped to Acc_1 (pink parts), so Acc_2 remains idle. (c) The well-balanced mapping scheme. l_1 and l_2 are located on Acc_1 and Acc_2 respectively. Acc_2 is better utilized, but the overall latency is long. (d) The mapping scheme after Balance Aware Re-mapping. After re-mapping, l_2 in (b) is moved to the grey part, and the optimal latency is achieved.

B. DRAM Budget Aware Mapping

In this procedure, we initially map G^*_{model} to $\{Acc_i\}$ with the following steps. First, we find unmapped nodes without predecessors and enumerate all the combinations of mapping these nodes on $\{Acc_i\}$ (steps 2-3). For example, in Fig. 3 (a), start with sub-operations 1.1 and 1.2, the mapping combinations include {1.1 to Acc1, 1.2 to Acc1}, {1.1 to Acc1, 1.2 to Acc2}, {1.1 to Acc2, 1.2 to Acc1}, and {1.1 to Acc2, 1.2 to Acc2}. Second, we check if the current DRAM budget is possible to accommodate the data for the rest training sub-operations for all the mapping candidates and only calculate the latency increase ΔLat for the mapping candidate that satisfies the DRAM budget (steps 4-5). ΔLat is the sum of the computation cost calculated by the performance model of $\{Acc_i\}$ and the data communication costs between two dependent sub-operations that are located on different accelerators. For example, when both 1.1 and 1.2 are allocated to Acc2, the DRAM budget in Acc2 cannot hold weights and immediate features for the sub-operations. Thus, we only calculate ΔLat under the other three mapping schemes. Third, the mapping candidate that results in the minimum ΔLat is selected (step 6). Shown in Fig. 3 (c), the mapping scheme {1.1 to Acc1, 1.2 to Acc2} leads to the minimum ΔLat. Finally, we remove the mapped nodes from G^*_{model} (i.e. remove 1.1 and 1.2 from unmapped nodes and start with 2.1 and 2.2) and also update the DRAM budget of Acc1 and Acc2 (steps 8-9).

As mentioned in Sec. II-D, training has more intricate data dependency and memory budget compared to MMMT inference. Therefore, we have the following improvements on top of H2H. Firstly, H2H is for inference which only transfers activations in FP, but training needs to consider different data transmission situations in FP/BP/WU. Therefore, we analyze the communication costs which are as follows. In FP, if two dependent layers (e.g. l_1 and l_3 in Fig. 1 (b) are located on different accelerators, only the immediate activation (i.e. the output features of l_1) of the predecessor needs to be transmitted to the successor's layer. Therefore, the communication cost is the size of A_2B_1 divided by the bandwidth between two accelerators. In BP, we take l_6 as an example. It not only receives output features L_4B_1 from l_4 but also receives weights from l_4. Thus, the communication costs include the immediate features from its predecessors that are located on different accelerators and the weights from its FP counterpart. In WU, e.g. l_{11}, it receives output features L_2B_1 from l_{10} and input features A_1B_1 from l_1 to calculate weights gradients dW_2B_1. Then, the gradients are scattered to all accelerators and accumulated to update the weights for FP in the next mini-batch training or for inference.

The second difference lies in the different DRAM costs between inference and training. In MMMT inference, the memory costs for an operation come from two aspects: weights and immediate activation. Weights are stored in DRAM where the accelerator is located for the whole inference process, while immediate activation can be eliminated when the successor operation finishes. However, in training, apart from weights, immediate activation and loss also need to be saved

until weight gradients are calculated. Therefore, during the mapping, we check the DRAM budget and only select the mapping candidates that satisfy the DRAM budget in step 4 and update the budget in step 8, which we call DRAM budget aware. In step 8, activation/loss is accumulated in local DRAM in FP/BP. If a WU operation is finished, we release the memory that stores its activation in FP and loss in BP. For example, we release the memory for L_2B_1 and A_1B_1 after l_{11} is completed.

C. Balance Aware Re-mapping

After we get the initial G^*_{sys} and Lat via DRAM Budget Aware Mapping, we balance the workload among accelerators to further improve the training performance. The pink boxes in Fig. 7 (b) represent an example of mapping a model in Fig. 7 (a) with DRAM Budget Aware Mapping. The powerful accelerator Acc_1 processes 4 sub-operations, while the low-end accelerator Acc_2 remains idle. Fig. 7 (c) assigns these sub-operations on the accelerators in a well-balanced manner, but the overall latency is longer due to the low performance of Acc_2. Therefore, we propose a Balance Aware Re-mapping approach combining the advantages of both Fig. 7 (b) and Fig. 7 (c). This optimization is innovative compared to H2H.

First, as illustrated in Alg. 2, steps 11-13, in the mapping scheme in Fig. 7 (b), start from unbalanced sub-operations without predecessors, we check if they are located on the balanced accelerators like Fig. 7 (c). Second, if such a sub-operation is founded, we try to move it to its balanced accelerator and update G^*_{sys} and Lat (steps 14-15). For example, in Fig. 7 (b), we move l_2 from Acc_1 to Acc_2 which is represented in the grey box. Third, if the overall latency is shortened, we accept this balanced scheme and update G^*_{sys} and Lat (steps 16-17). Finally, we move to the next unbalanced sub-operation. The Balance Aware Re-mapping stops when all the sub-operations are checked (step 19). The mapping scheme after Balance Aware Re-mapping is shown in Fig. 7 (d).

D. Data Locality Aware Re-mapping

This section further reduces the overall latency by re-allocating a sub-operation from its source accelerator to a new destination accelerator, on which its predecessors and/or successors are mapped. Such an approach can reduce feature transmission latency [34].

First, for each node in G^*_{model}, if its predecessor or successor is not on the same accelerator that the node is located, we attempt to re-map the node to its predecessor's/successor's accelerator (steps 22-23). Compared to H2H, the predecessors and successors are more complicated in FP/BP/WU. In this work, we consider all the data transfer relationships mentioned in Sec. V-B. Second, for each re-mapping attempt, we calculate the overall latency after re-mapping (step 24). Third, if the overall model latency is shortened, we accept the current re-mapping attempt (steps 25-26). If no more layers can be re-mapped with reduced training latency, the Data Locality Aware Re-mapping stops (step 28).

As introduced in Sec. II-B, current FPGA-based training accelerators are only designed for CNN operations, while operations like attention or transformer have not been explored. Besides, as introduced in Sec. I, our goal aims to scale FPGA-based edge servers to on-device training tasks. Those servers, such as VMSS, are currently targeting CNNs for video processing tasks. Although our work currently targets CNN models, it can also be extended to other modern models as long as correspondent operations are supported by new training accelerators. For example, if FPGA-based training accelerators for attention or transformers are invented with accurate profiling models, same as CNNs, these operations are first categorized as batch-wise operations or non-batch-wise operations. In MTrain-Converting, we partition non-batch-wise operations just like Conv operations. For batch-wise operations, the same as BN, data from its predecessor layers are fused from all devices. The converted graph is exactly the same as G^*_{model} in MTrain-Mapping, so Alg. 2 can also be applied. The only differences are as follows. First, in step 3, attention operations can only mapped to attention accelerator candidates, while Conv operations can only be mapped to Conv accelerators. Second, in step 14, we re-map when n has the same operation type as the target accelerator. Last, in step 23, re-mapping is only considered when n has the same operation type as its predecessor or successor. Same with transformer or attention mechanisms, mixed-domain models can also be supported as long as training accelerators for all domain models are well developed.

VI. Experiments

In this section, we first analyze and show the effectiveness of re-mapping and re-partition in MTrain. Secondly, we present the overall effectiveness of MTrain by comparing it with SOTA training algorithms. Thirdly, we present a series of ablation studies to validate that MTrain-Converting can efficiently achieve near-optimal solutions in an acceptable time. Finally, we analyze the throughput and complexity when edge servers are scaled to larger edge clusters. All these experiments use throughput to evaluate the training performance. It should be noted that using throughput as the metric is equal to using energy efficiency in the following experiments. Given a CNN model, for each FPGA in the cluster, we first select accelerator IPs that have the maximum throughput and deploy them on the device. Then, we apply different mapping algorithms including the proposed MTrain and SOTA baselines for optimal throughput. In these FPGA-based training accelerator designs [13], [15], once the accelerator IP is implemented on the FPGA with bitstream generation, the estimated power consumption is fixed. Therefore, in each experiment, all counterparts share the same power consumption. Energy efficiency is throughput divided by power consumption. Divided by the same ratio, the trend of energy efficiency is the same as throughput.

A. Experimental Setup

Heterogeneous FPGA-based Edge Clusters. Table II summarizes 3 heterogeneous edge clusters that are used in FPGA-based inference works. VMSS is composed of Xilinx U50LV and U30 equipped with PCIe [3]. BLAST-R employs high-end (XCZU9EG), mid-end (XC7Z045), and low-end (XC7Z015)

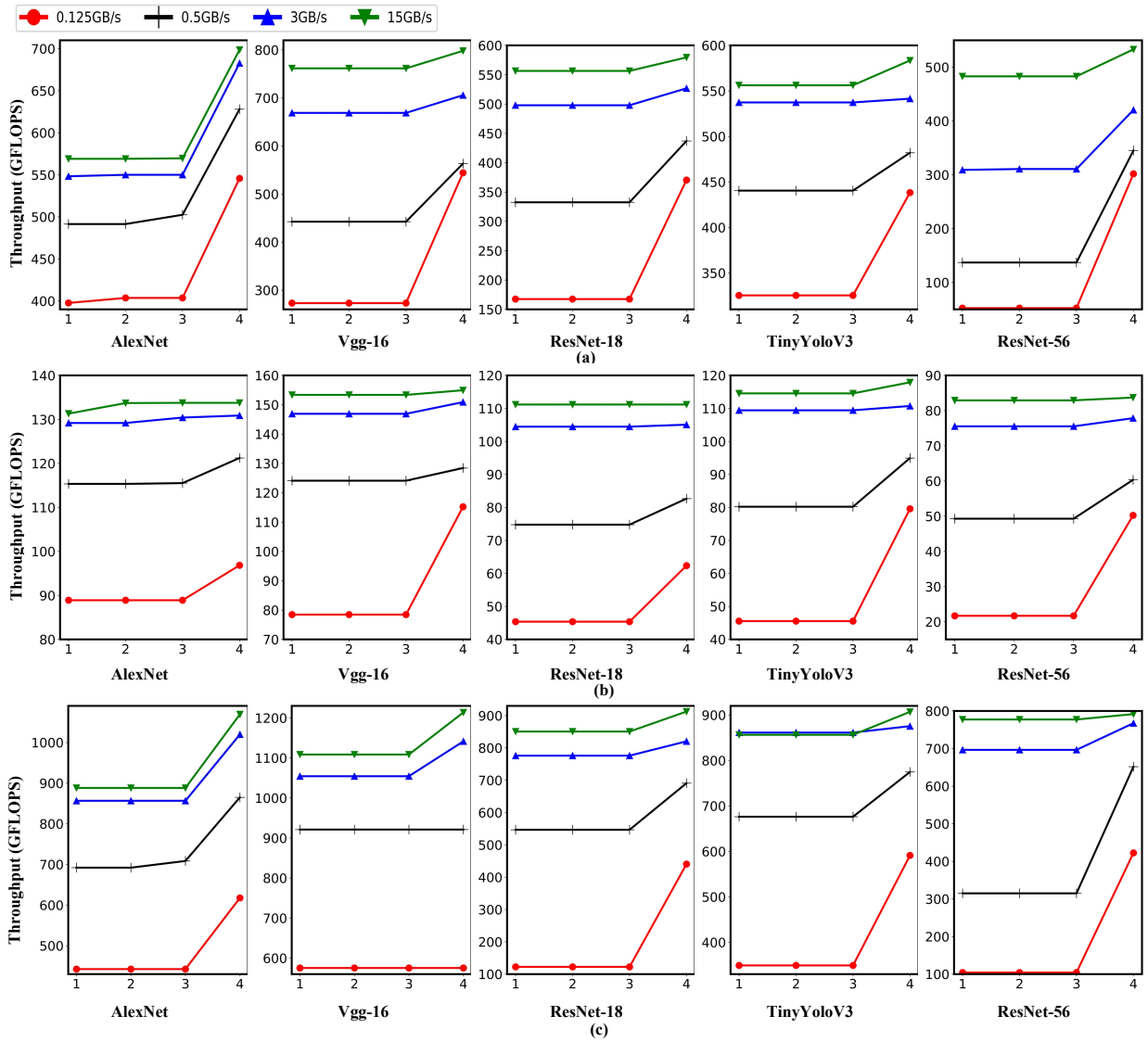

Fig. 8. The throughput comparisons for different optimization steps under different P2P communication bandwidth configurations. The X-axis represents the following steps: step 1 is after PE-based Initial Partition and DRAM Budget Aware Mapping with the initial ratio, step 2 is after Balance Aware Re-mapping with the initial partition ratio, step 3 is after Data Locality Aware Re-mapping with the initial ratio, and step 4 is after Duty-cycle-based Re-partition and MTrain-Mapping with the final ratio. (a) Comparisons for Cluster 1. (b) Comparisons for Cluster 2. (c) Comparisons for Cluster 3. The re-mapping and re-partition can effectively improve the training throughput for the clusters.

Xilinx FPGAs, and the FPGAs are connected with high-speed serial (HSS) [5]. In [36], VCU128, ZCU102, and ZCU104 are connected to a high-speed switch for communication through Ethernet ports. To show the training performance on various P2P communication bandwidths, we test MTrain on these clusters with bandwidth ranging from 0.125 GB/s to 15 GB/s. The training batch size for Cluster 1 and Cluster 3 is 64, while that for Cluster 2 is 16 due to the DRAM size constraint.

TABLE II
HETEROGENEOUS EDGE CLUSTERS

Name	Used in	Configuration
Cluster 1	VMSS [3]	U50LV+U30
Cluster 2	BLAST-R [5]	XCZU9EG+XC7Z045+XC7Z015
Cluster 3	[36]	VCU128+ZCU102+ZCU104

Training Accelerator IPs. We survey SOTA FPGA-based training accelerators and select DarkFPGA [13] and EF-train [15] which have established complete performance and resource models in FP/BP/WU with end-to-end implementation validation. We replicate the models based on the original papers. Both accelerators adopt 32-bit floating-point.

Modern CNNs. We use AlexNet [37], Vgg-16 [30], ResNet-18 [31], TinyYoloV3 [32], and ResNet-56 [38] to evaluate the effectiveness of MTrain. Vgg-16, ResNet-18, TinyYoloV3, and ResNet-56 include BN operations (Feature ④), while ResNet-18, TinyYoloV3, and ResNet-56 have cross-branch data dependency (Feature ③).

Baselines.

- To show the effectiveness of re-mapping and re-partition optimizations in Sec. VI-B, we compare the throughput after the following steps: Step 1 represents the training throughput when MTrain-Converting provides PE-

based Initial Partition and MTrain-Mapping conducts DRAM Budget Aware Mapping. Step 2 represents the throughput after Balance Aware Re-mapping with the initial partition ratio. Step 3 represents throughput after Data Locality Aware Re-mapping under the same initial ratio. Step 4 represents the final throughput, i.e. MTrain-Mapping conducts DRAM Budget Aware Mapping, Balance Aware Re-mapping, and Data Locality Aware Re-mapping with the final partition ratio after Duty-cycle-based Re-partition.

- Current multi-accelerator training optimization methods are more focused on cloud-level servers like GPUs while training on edge clusters has not been well investigated. To show the overall effectiveness of MTrain in edge clusters, in Sec. VI-C, we compare it with two SOTA multiple accelerator training algorithms: the ILP-based sharding algorithm in Alpa and the optimization algorithm in HAP that jointly finds the best sharding scheme and the sharding ratio. Alpa and HAP achieve high throughput in high-performance computing cloud with abundant GPU nodes and high P2P bandwidth. Alpa targets a homogeneous Amazon EC2 cluster of 8 p3.16xlarge instances with 64 GPUs, while HAP targets a public heterogeneous cloud with 64 GPUs in total. We apply the scheduling algorithms of Alpa and HAP on our resource-constrained edge servers and compare the training performance with MTrain. We also compare MTrain with single FPGA training accelerators EF-Train and DarkFPGA targeting on edge devices.
- To show the efficiency and effectiveness of MTrain-Converting in Sec. VI-D, we compare the resultant throughput and search time with the optimal solution generated by depth-first search (DFS).

B. Effectiveness of Re-mapping and Re-partition

The training throughput of the three clusters listed in Table II under different P2P bandwidths is illustrated in Fig. 8. The X-axis represents different optimization steps in MTrain.

From Fig. 8, we can have the following observations. Firstly, it can be seen that the proposed re-mapping and re-partition techniques work jointly and effectively improve the training throughput. For example, in Fig. 8 (a), when the P2P bandwidth is 0.125 GB/s, the throughput after step 4 increases by 37% compared to the throughput in step 1 for AlexNet. Secondly, different P2P bandwidth configurations have a significant impact on the overall training throughput. For example, for AlexNet, the final throughput increases by 28% in Cluster 1 when the bandwidth ranges from 0.125 GB/s to 15 GB/s. Thirdly, the overall throughputs of ResNet-18 and TinyyoloV3 are lower than that of AlexNet and Vgg-16. For example, in Cluster 1, the maximum throughput for ResNet-18 is 580 GFLOPS, the maximum throughput for TinyYoloV3 is 583 GFLOPS, the maximum throughput for AlexNet is 699 GFLOPS, and the maximum throughput for Vgg-16 is 798 GFLOPS. The performance in Clusters 2 and 3 has the same trends. For example, in Cluster 2, the throughput for Vgg-16 is 128 GFLOPS under 0.5 GB/s BW, while the throughput of ResNet-18 is 83 GB/s BW, and the throughput for TinyYoloV3 is 95 GB/s. In Cluster 3, the throughput for Vgg-16 is 921 GFLOPS under 0.5 GB/s BW, while the throughput of ResNet-18 is 691 GFLOPS, and that for TinyYoloV3 is 774 GFLOPS. This is because the operations in the former models are less computationally intensive compared to the latter ones. For example, layers in ResNet-18 have fewer filters and lower complexity than Vgg-16 [31], and each training accelerator benefits computation-intensive operations since the communication costs can be covered by the computation costs. Besides, the cross-data dependency in ResNet-18 or TinyYoloV3 causes more inter-FPGA communication overhead. The only violation is that in Cluster 3, the throughput of TinyYoloV3 is 591 GFLOPS under 0.125 GB/s BW which is slightly higher than 575 GFLOPS of Vgg-16. It is because Re-partition under low BW like 0.125 GB/s aims to avoid communication overhead by centralizing workloads on one device rather than distributing them to all devices. In Cluster 3, Re-partition works more effectively for TinyYoloV3 than Vgg-16.

Currently, we provide pre-computed mapping solutions for static communication bandwidth. If bandwidth varies, we just use the average bandwidth based on its device-to-device communication approach. It should be noted that our algorithm can also be extended to support dynamic changing communication under the following two situations. First, suppose we still generate a pre-computed mapping solution before training. In that case, we will measure bandwidth variations of the cluster during the previous training epochs and then divide the variations into different ranges. When the bandwidth fluctuates under a given range, we use the MTrain mapping result under the average value of such range. Second, if a run-time mapping is required in the training system, at time t_0, we can apply the mapping result of average bandwidths during $[t_0 - T,\ t_0]$, where T is a hyper-parameter determined by the search time of MTrain and bandwidth fluctuation frequency.

C. Comparison with SOTA Works

In this section, we compare the training throughput of our proposed MTrain with Alpa and HAP to test the overall effectiveness of our work. As explained in Sec. III, the training scheduling algorithm provides a pre-computed optimal solution. Therefore, the search time of SOTA and our work is excluded from the throughput evaluation. The results for Cluster 1, Cluster 2, and Cluster 3 are shown in Fig. 9, 10, and 11, respectively. The X-axis represents different P2P bandwidths. The proposed MTrain effectively solves the computation complexity and communication bottlenecks compared to SOTA works.

Although Alpa and HAP achieve high throughput in cloud servers, they cannot work well in resource-constrained edge servers. As shown in Figs. 9-11, MTrain outperforms Alpa and HAP for all three clusters. Compared with the most recent work, HAP, MTrain achieves 1.07x-2.21x speedup under 15 GB/s P2P bandwidth. Alpa and HAP achieve worse performance when the P2P bandwidth is lower. This is because Alpa and HAP distribute each training operation on all devices without considering the communication overhead. When the

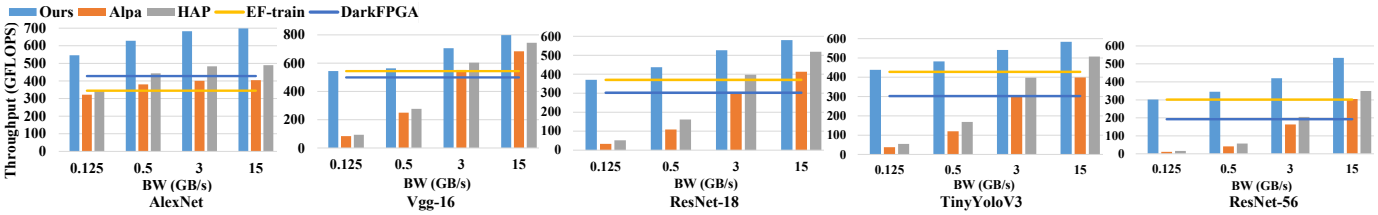

Fig. 9. The throughput comparison with SOTA works for Cluster 1, and the single mapping strategy targets U50LV of Cluster 1.

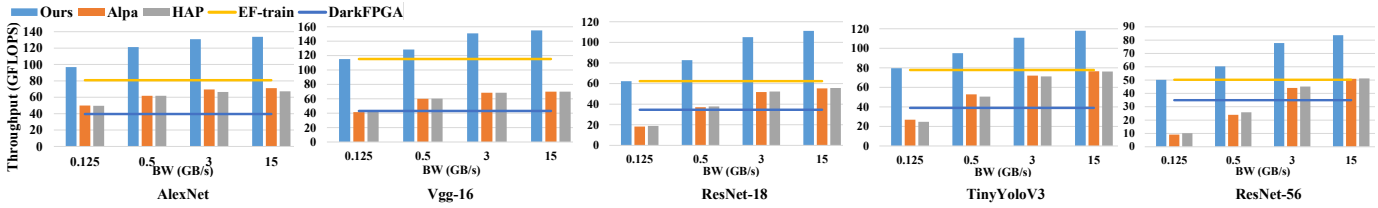

Fig. 10. The throughput comparison with SOTA works for Cluster 2, and the single mapping strategy targets XCZU9EG of Cluster 2.

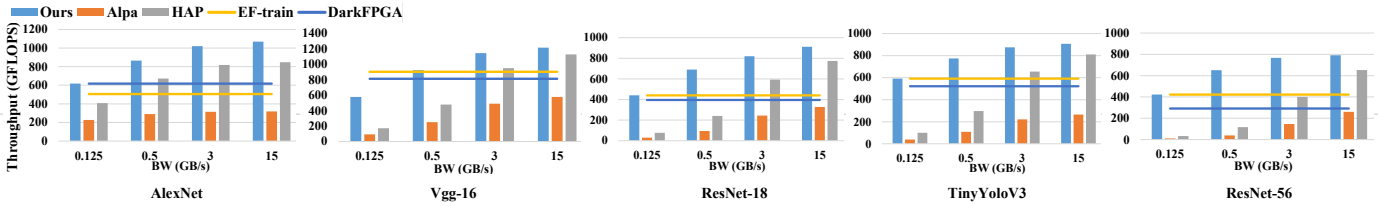

Fig. 11. The throughput comparison with SOTA works for Cluster 3, and the single mapping strategy targets VCU128 of Cluster 3.

P2P bandwidth is low, accelerating the training process on more accelerators cannot compensate for the communication bottleneck. To further show the impact of the communication bottleneck, we also apply a single mapping strategy, i.e. only scheduling the training process on the most resource-abundant FPGA of a given cluster.

The blue and yellow lines in Figs. 9-11 show the training throughput scheduling on single accelerator DarkFPGA [13] and EF-train [15]. The accelerators are deployed on only one FPGA of the cluster, respectively. Compared to Alpa and HAP, when the P2P bandwidth is lower, training on a single device is optimal rather than distributing the workload to every FPGA. When the bandwidth increases to 15 GB/s, distributing the computation workload on more accelerators can compensate for the communication bottleneck.

MTrain is capable of balancing the trade-off between distributing the computation workload on multiple accelerators and avoiding communication bottlenecks. As shown in Figs. 9-11, when the bandwidth is lower, MTrain tends to generate a scheduling scheme similar to the single mapping strategy. When the bandwidth is higher, MTrain distributes training operations on multiple FPGAs. The better accelerator utilization and reduced communication costs introduced in Sec. III-C enable MTrain to outperform Alpa and HAP.

D. The Effectiveness and Efficiency of MTrain-Converting

As mentioned in Sec. IV, it is time-consuming to find an optimal partition ratio on an N-accelerator cluster when the batch size is B, so we propose MTrain-Converting to search for a near-optimal solution. In this section, we validate the effectiveness and efficiency of MTrain-Converting by comparing the resultant throughput and search time with the optimal solution. These results show that MTrain-Converting can achieve near-optimal solutions with significantly less search time.

TABLE III
THE MODEL CONVERTING PERFORMANCE AND SEARCH TIME COMPARISON

		DFS (Optimal)		MTrain-Converting (ours)	
Model	Cluster	Thp. (GFLOPS)	ST (s)	Thp. (GFLOPS)	ST (s)
AlexNet	1	683	27.55 (3.63x)	683 (1x)	7.58
	2	138	220.57 (15.62x)	131 (0.95x)	14.12
Vgg-16	1	706	1099.64 (4.10x)	706 (1x)	268.19
	2	157	1079.38 (15.07x)	151 (0.96x)	71.64
ResNet-18	1	527	905.93 (4.16x)	527 (1x)	217.52
	2	108	129009.68 (24.81x)	105 (0.97x)	5200.76
TinyYoloV3	1	541	716.32 (4.07x)	541 (1x)	176.05
	2	114	35559.63 (24.55x)	111 (0.97x)	1448.61

To find the optimal solution, we first use DFS to enumerate all possible partition ratios. Then, we conduct MTrain-Mapping for each ratio and find the best ratio that leads to the maximal throughput after mapping. For Cluster 1 with $B = 64$ and $N = 2$, there are 33 feasible ratios. For Cluster 2 with $B = 16$ and $N = 3$, there are 33 possible ratios. For Cluster 3 with $B = 64$ and $N = 3$, there are 385 possible ratios. When the number of accelerators increases, the search time for MTrain-Mapping also increases. For example, mapping ResNet-18 on Cluster 1 under the initial PE-based partition

ratio costs only 3.07s. Mapping ResNet-18 on Clusters 2 and 3 takes up 5071s and 4994s, respectively. Therefore, finding an optimal solution for Cluster 3 is estimated to cost around 22 days, which is time-consuming and inefficient in practical applications. Therefore, we compare MTrain-Converting with the optimal solution for Cluster 1 and Cluster 2.

Table III compares the throughput and search time under 3 GB/s p2p bandwidth. Our proposed MTrain-Converting can achieve near-optimal performance for all the scenarios, while searching for an optimal partition ratio demands a great mass of time. For example, MTrain-Converting schedules ResNet-18 on Cluster 2 and achieves 105.07 GFLOPS throughput, which is 97% compared to the optimal solution. However, MTrain-Converting costs only 1.44h to find the near-optimal solution, while the optimal solution costs around 36h, 24.81 times compared to MTrain-Converting.

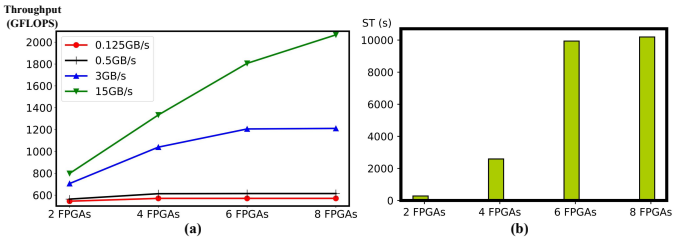

Fig. 12. Scalability of MTrain when the number of FPGAs increases from 2 to 8. (a) Throughput under different P2P BW. (b) Search time of MTrain.

E. Scalability Analysis

Above mentioned experiments have shown the efficiency and effectiveness of MTrain in practical edge servers. It is also promising to scale on-device training to larger edge clusters in the future. In this section, we use the VMSS configuration and scale the nodes from 2 FPGAs (i.e. one pair of U50 and U30) to 8 FPGAs (i.e. 4 pairs of U50 and U30). It should be noted that current edge-level clusters mainly use less than 4 FPGAs, so clusters with 8 nodes are large enough. The overall throughput is shown in Fig. 12 (a). It can be shown that when the P2P bandwidths are low, e.g. under 0.5 GB/s, increasing FPGA nodes cannot improve the overall performance. This is because the communication bottleneck under such bandwidths plays a more important role. Rather than distributing the workload to more devices, centralizing the whole training process is the best choice. Under higher bandwidth, e.g. 15 GB/s, the throughput significantly improves when the FPGAs increase.

We also analyze the complexity using the overall search time of MTrain. Fig. 12 (b) illustrates the search time under 3 GB/s P2P bandwidth. When the FPGAs increase to 8, the overall search time is around 2.83h, which is acceptable in the on-device learning scenarios described in Sec. III-A.

VII. Conclusion

This work proposes MTrain to enable efficient CNN training on heterogeneous FPGA-based edge servers. It transfers the training process into a multi-branch workflow with independent sub-operations on different branches and allocates each sub-operation on different training accelerators for better utilization and reduced communication overhead. It achieves high training throughput on resource-constrained edge servers compared with SOTA multi-accelerator training works.

References

[1] "Smart City with VMSS," https://www.xilinx.com/video/application/smart-city-with-vmss.html.

[2] "Smart City Edge Server," https://www.xilinx.com/products/intellectual-property/1-1971pd6.html.

[3] "Bring Video Analytics to a new level with Xilinx's VMSS and VCK5000 Platform," https://www.xilinx.com/developer/articles/video-analytics-with-vmss-and-vck5000.html.

[4] P. Szántó, T. Kiss, and K. J. Sipos, "Fpga accelerated deepsort object tracking," in *2023 24th International Carpathian Control Conference (ICCC)*. IEEE, 2023, pp. 423–428.

[5] W. Jiang, E. H.-M. Sha, Q. Zhuge, L. Yang, X. Chen, and J. Hu, "Heterogeneous fpga-based cost-optimal design for timing-constrained cnns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2542–2554, 2018.

[6] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kb memory," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 941–22 954, 2022.

[7] Z. Zhang, D. Zhao, R. Liu, K. Tian, Y. Yao, Y. Li, and H. Ma, "Camonet: On-device neural network adaptation with zero interaction and unlabeled data for diverse edge environments," *IEEE Transactions on Mobile Computing*, 2024.

[8] X. Qiu, J. Fernandez-Marques, P. P. Gusmao, Y. Gao, T. Parcollet, and N. D. Lane, "Zerofl: Efficient on-device training for federated learning with local sparsity," *arXiv preprint arXiv:2208.02507*, 2022.

[9] H. Cho, A. Mathur, and F. Kawsar, "Flame: Federated learning across multi-device environments," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 3, pp. 1–29, 2022.

[10] J. Liu, X. Yu, and T. Rosing, "Self-train: Self-supervised on-device training for post-deployment adaptation," in *2022 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 2022, pp. 161–168.

[11] B. Li, Y. Lin, and I. A. Khan, "Self-supervised learning iot device features with graph contrastive neural network for device classification in social internet of things," *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 4255–4267, 2023.

[12] T.-J. Yang, D. Guliani, F. Beaufays, and G. Motta, "Partial variable training for efficient on-device federated learning," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 4348–4352.

[13] C. Luo, M.-K. Sit, H. Fan, S. Liu, W. Luk, and C. Guo, "Towards efficient deep neural network training by fpga-based batch-level parallelism," *Journal of Semiconductors*, vol. 41, no. 2, p. 022403, 2020.

[14] S. K. Venkataramanaiah, H.-S. Suh, S. Yin, E. Nurvitadhi, A. Dasu, Y. Cao, and J.-s. Seo, "Fpga-based low-batch training accelerator for modern cnns featuring high bandwidth memory," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–8.

[15] Y. Tang, X. Zhang, P. Zhou, and J. Hu, "Ef-train: Enable efficient on-device cnn training on fpga through data reshaping for online adaptation or personalization," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 5, pp. 1–36, 2022.

[16] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 37, no. 1, pp. 35–47, 2017.

[17] F. Liu, W. Zhao, Z. Wang, Y. Zhao, T. Yang, Y. Chen, and L. Jiang, "Ivq: In-memory acceleration of dnn inference exploiting varied quantization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5313–5326, 2022.

[18] H.-Y. Chiang, N. Frumkin, F. Liang, and D. Marculescu, "Mobiletl: on-device transfer learning with inverted residual blocks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 6, 2023, pp. 7166–7174.

[19] L. Song, F. Chen, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Accpar: Tensor partitioning for heterogeneous deep learning accelerators," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 342–355.

[20] "Amazon EC2," https://aws.amazon.com/ec2/instance-types/.
[21] "NVLink and NVSwitch," https://www.nvidia.com/en-us/data-center/nvlink/.
[22] "The Heterogeneous Accelerated Compute Cluster at the University of Illinois, Urbana-Champaign," https://xilinx-center.csl.illinois.edu/xacc-cluster/.
[23] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing *et al.*, "Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 559–578.
[24] T. Wang, T. Geng, A. Li, X. Jin, and M. Herbordt, "Fpdeep: Scalable acceleration of cnn training on deeply-pipelined fpga clusters," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1143–1158, 2020.
[25] P. Kreowsky, J. Knapheide, and B. Stabernack, "An approach towards distributed dnn training on fpga clusters."
[26] C. Jiang, Z. Jia, S. Zheng, Y. Wang, and C. Wu, "Dynapipe: Optimizing multi-task training through dynamic pipelines," *arXiv preprint arXiv:2311.10418*, 2023.
[27] S. Zhang, L. Diao, C. Wu, Z. Cao, S. Wang, and W. Lin, "Hap: Spmd dnn training on heterogeneous gpu clusters with automated program synthesis," *arXiv preprint arXiv:2401.05965*, 2024.
[28] P. Liang, Y. Tang, X. Zhang, Y. Bai, T. Su, Z. Lai, L. Qiao, and D. Li, "A survey on auto-parallelism of large-scale deep learning training," *IEEE Transactions on Parallel and Distributed Systems*, 2023.
[29] V. Thakkar, S. Tewary, and C. Chakraborty, "Batch normalization in convolutional neural networks—a comparative study with cifar-10 data," in *2018 fifth international conference on emerging applications of information technology (EAIT)*. IEEE, 2018, pp. 1–5.
[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[31] 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.
[32] P. Adarsh, P. Rathi, and M. Kumar, "Yolo v3-tiny: Object detection and recognition using one stage improved model," in *2020 6th international conference on advanced computing and communication systems (ICACCS)*. IEEE, 2020, pp. 687–694.
[33] C. Hao and D. Chen, "Software/hardware co-design for multi-modal multi-task learning in autonomous systems," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2021, pp. 1–5.
[34] X. Zhang *et al.*, "H2h: Heterogeneous model to heterogeneous system mapping with computation and communication awareness," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 601–606.
[35] A. K. Kamath *et al.*, "M5: Multi-modal multi-task model mapping on multi-fpga with accelerator configuration search," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.
[36] W. Teng, L. Gong, C. Wang, and X. Zhou, "Enabling elastic resource management in cloud fpgas via a multi-layer collaborative approach," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2023, pp. 242–244.
[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.
[38] J. Wu, D. Zhu, L. Fang, Y. Deng, and Z. Zhong, "Efficient layer compression without pruning," *IEEE Transactions on Image Processing*, 2023.

VIII. Biography Section

Yue Tang is with the University of Pittsburgh, Pittsburgh, USA. She received her Ph.D. degree in electrical and computer engineering from the University of Pittsburgh, USA, in 2024. She received her B.S. and M.S. degrees from the School of Automation Science and Electrical Engineering, Beihang University, China, in 2016 and 2019, respectively. Her research interests include FPGA-based CNN training and on-device AI. She has received the 2023 ACM TODAES Rookie Author of the Year (RAY) Award.

Alex K. Jones is the Klaus Schroder Endowed Chair Professor of Engineering and Computer Science and Department Chair of EECS (Electrical Engineering and Computer Science) at Syracuse University. Previously he was a Professor at the University of Pittsburgh and served in a variety of roles at the NSF including Program Director in CISE/CNS and Deputy Division Director of ENG/ECCS. He received his M.S. and Ph.D, in Electrical and Computer Engineering at Northwestern University where he was a Walter P. Murphy Fellow. Dr. Jones is known for advancing the field of sustainable computing with full lifecycle carbon modeling and optimization. His research interests include fault tolerance and computing in conventional and emerging memories, compilation for configurable systems and architectures, and quantum computing. He is the steering committee chair for the IEEE International Green and Sustainable Computing Conference and has served on program committees of ISCA, MICRO, HPCA, DAC among others. He is a topical editor for the IEEE Transactions on Computers, and an associate editor for the IEEE Transactions on Sustainable Computing.

Jinjun Xiong (F'23) received his Ph.D. degree from the University of California Los Angeles (UCLA), USA, in 2006. He is currently an Empire Innovation Professor with the Department of Computer Science and Engineering at University at Buffalo (UB). He is also the Director of UB's Institute for Artificial Intelligence and Data Science (IAD), the Scientific Director for the National Artificial Intelligence Institute for Exceptional Education (AI4ExceptionalEd), and the AI Thrust Lead for the National Center for Early Literary and Responsible AI (CELaRAI). Prior to UB, he was a Senior Researcher and Program Director for AI and Hybrid Clouds Systems at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA. His research interests are in across-stack AI systems research, including AI applications, algorithms, tooling, and computer architectures. Many of his research results have been adopted in industrial products and tools. His publication has won 9 Best Paper Awards and 10 Nominations for Best Paper Awards at various international conferences.

Peipei Zhou is currently an Assistant Professor at the School of Engineering, Brown University. She received her Ph.D. in Computer Science (2019) and M.S. in Electrical and Computer Engineering (2014) from UCLA, and her B.S. in Electrical and Computer Engineering (2012) from Southeast University. Her research investigates architecture, programming abstraction, and design automation tools for reconfigurable computing and heterogeneous computing. She has published papers in IEEE/ACM computer system and design automation conferences and journals. Her work has received 2 best paper awards, including 2019 IEEE TCAD Donald O. Pederson Best Paper Award and 4 best paper nominations. She serves as an associate editor for IEEE Transactions on Computers and ACM Transactions on Reconfigurable Technology and Systems.

Jingtong Hu is currently an Associate Professor in the Department of Electrical and Computer Engineering at University of Pittsburgh, Pittsburgh, PA, USA and a consultant for nonprofit organization One Heart Health (1HH). He received his Ph.D. in Computer Science from the University of Texas at Dallas in 2013 and his B.E. in Computer Science and Technology from Shandong University, China in 2007. His research interests include embedded systems, on-device AI, and digital health, with a focus on achieving independent, personalized, and inclusive AI-powered healthcare systems through hardware/software co-design. His works have received 3 best paper awards, including the Donald O. Pederson Best Paper Award from IEEE Transactions on Computer-Aided Design of Circuits and Systems, and 5 best paper nominations. He is also the recipient of DAC Under-40 Innovators Award, JSPS Invitational Fellowships for Research, Germany Humboldt Research Fellowship, and ACM SIGDA Meritorious Service Award. He has served on the technical program committee of many international conferences. He also served as an executive committee member and education chair for ACM SIGDA and associate editor for IEEE Embedded Systems Letters, the Journal of Systems Architecture: Embedded Software Design, ACM Transactions on Design Automation of Electronic Systems, and ACM Transactions on Cyber-Physical Systems.