TrainBox: An Extreme-Scale Neural Network Training Server Architecture by Systematically Balancing Operations

Pyeongsu Park Heetaek Jeong Jangwoo Kim*
Department of Electrical and Computer Engineering
Seoul National University
{pyeongsu, heetaek, jangwoo}@snu.ac.kr

***Abstract*— Neural network is a major driving force of another golden age of computing; the computer architects have proposed specialized accelerators (e.g., TPU), high-speed interconnects (e.g., NVLink), and algorithms (e.g., ring-based reduction) to efficiently support neural network applications. As a result, they achieve orders of magnitude higher efficiency for neural network computation and inter-accelerator communication over traditional computing platforms.**

In this paper, we identify that the emerging platforms have shifted the performance bottleneck of neural network from model computation and inter-accelerator communication to *data preparation*. Although overlapping data preparation and the others has hidden the preparation overhead, the higher input processing demands of emerging platforms start to reverse the situation; at scale, data preparation requires an infeasible amount of the host-side CPU, memory, and PCIe resources. Our detailed analysis reveals that this heavy resource consumption comes from data transformation for neural network specific formats, and buffering for communication among devices.

Therefore, we propose a scalable neural network server architecture by balancing data preparation and the others. To achieve extreme scalability, our design relies on a scalable device array, rather than the limited host resources, with three key ideas. First, we offload CPU-intensive operations to the customized data preparation accelerators to scale the training performance regardless of the host-side CPU performance. Second, we apply direct inter-device communication to eliminate unnecessary data copies and reduce the pressure on the host memory. Lastly, we cluster underlying devices considering unique communication patterns of the neural network processing and interconnect characteristics to efficiently utilize aggregated interconnect bandwidth. Our evaluation shows that the proposed architecture achieves 44.4× higher training throughput on average over a naively extended server architecture with 256 neural network accelerators.

***Index Terms*—Neural network, Training, Scalability, Data preparation, Server architecture**

I. Introduction

A new golden age driven by neural network creates many new opportunities for human society. For instance, neural network has been replacing many labor-intensive tasks into fully-/semi-automatic processes (e.g., autonomous vehicle [4]).

Before we use neural network models, we need to *train* them, which involves data preparation, model computation, and model synchronization regardless of input types and neural network models (Figure 1); We first prepare a data batch, then feed it to a model to calculate parameter adjustment factors. After computation on each device, we synchronize the result through model synchronization to provide a synchronized view of the model. Training repeats these steps for a large amount of data to get a high-quality neural network model.

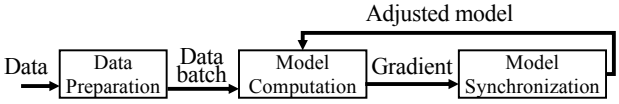
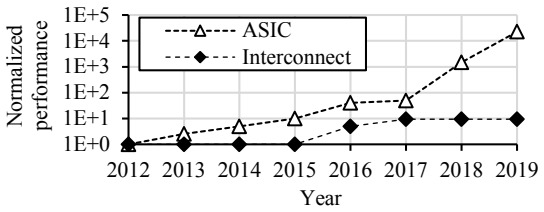
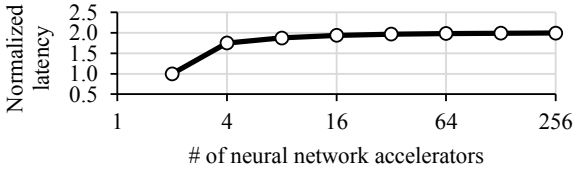


Fig. 1: Simplified neural network training steps.



(a) Performance (throughput/power) trends of hardware components for neural network [2], [5], [6], [11], [21], [27], [29], [33], [47].



(b) Model synchronization latency of a 4-KB-chunked ring (normalized to the synchronization latency with two accelerators) [25].

Fig. 2: Summary of hardware and software trends

As neural network requires massive computation for a large amount of data, training is extremely compute-intensive and challenging. Among the steps, model computation and synchronization have been the major interests because their overhead typically has been higher than the data preparation overhead. In addition, since we can overlap data preparation for the next batch with the computation and synchronization for the current batch, the data preparation overhead has been hidden or ignored most of the time. Therefore, hardware

*Corresponding author.

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00072

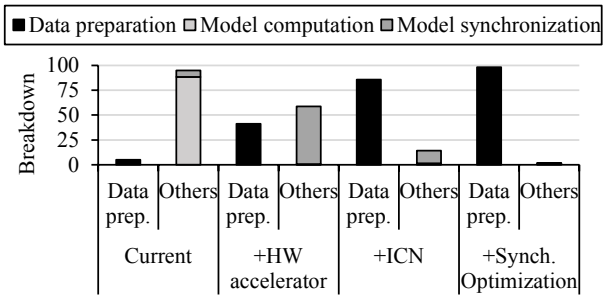


Fig. 3: Latency decomposition of an example neural network model (Resnet-50) with various optimizations (Data prep. + Others = 100%). Current: 8 Titan XP GPUs with PCIe Gen3, HW accelerator: 256 TPU v3-8, ICN: NVLink-speed interconnect, Synch. Optimization: Ring-based reduction.

designers have been seeking for efficient platforms for model computation and synchronization (Figure 2a). For example, a recently proposed accelerator shows more than 10,000× higher computation efficiency (i.e., throughput/watt) than the neural network accelerator in 2012.

At the same time, system architects are developing many-accelerator systems to increase the computation capability beyond a single accelerator [13], [16], [20]. As the size of training datasets (i.e., the number of items and individual item size) and the spectrum of hyperparameters (e.g., model structure, learning rate) increase, AI practitioners start to demand a higher number of neural network accelerators. For this, the key challenge is to minimize the synchronization overhead among accelerators. To mitigate the expensive inter-node communication, each server equips many accelerators with high-speed intra-node interconnects (Figure 2a). Also, the ring-based communication becomes common to guarantee scalable synchronization by exploiting intrinsic communication patterns in training (Figure 2b). Consequently, each server can have a large number of accelerators with minimal performance degradation (e.g., NVIDIA's DGX-2 with 16 GPUs [33]).

However, we observe that the performance bottleneck will change with large-scale neural network servers (Figure 3). Because of overlapping (a) data preparation for the next batch and (b) model computation and synchronization for the current batch, the longest step between (a) and (b) becomes the performance bottleneck. While the bottlenecks of the traditional servers have been in (b), *our profiling shows that data preparation becomes a critical performance bottleneck at large scale.* As we apply hardware and software optimizations from left to right, data preparation starts to dominate overall latency. With 256 neural network accelerators and high-speed interconnects, data preparation shows 54.9× longer latency than the others in this example.

Our detailed profiling reveals that the performance bottleneck stems from the limited amount of host-side resources for data preparation. Specifically, data preparation requires many CPU cores and memory bandwidth for transforming input into neural network specific formats as well as augmentation, an important technique to increase model accuracy. In addition, as the data providers are storage devices and the consumers are neural network accelerators, the data transfer among them requires high memory bandwidth for temporal buffering and PCIe bandwidth for data movements. *For example, a server with 256 neural network accelerators requires, on average, 50.0×, 7.6×, and 7.1× more CPU cores, memory bandwidth, and PCIe bandwidth, respectively*, over high-end NVIDIA DGX-2. Furthermore, the problem will become worse for the next generation of neural network accelerators, interconnects, and emerging complex data preparation algorithms.

To address the system-level bottlenecks and scale training throughput to the extreme, we propose **TrainBox**, a neural network server architecture for a large number of neural network accelerators. We redesign the datapath of a neural network server by deploying a scalable device array, rather than relying on the limited host resources, to achieve extreme scalability with three key ideas. First, we offload CPU-heavy operations to an array of hardware accelerators to free the host-side CPU resources. Second, we rely on the peer-to-peer communication not to involve the host-side memory during data transfers. Lastly, TrainBox clusters devices considering the intrinsic communication patterns in data preparation, so that the server exploits the aggregate PCIe bandwidth rather than making a single-point hotspot at the PCIe root complex.

Our evaluation results show that TrainBox achieves much higher scalability than the naively extended DGX-2 style baseline with 256 neural network accelerators (44.4× higher throughput on average[1]).

Overall, this paper makes the following contributions:

- **Data preparation acceleration.** To the best of our knowledge, we are the first to enlighten the importance of data preparation for neural network servers at scale.
- **Important design guidelines.** We provide four design guidelines to enable scalable neural network servers. These design guidelines are applicable to a broad spectrum of neural network models and input types.
- **Novel TrainBox architecture**. TrainBox effectively removes the performance bottleneck caused by the limited host-side resources, and achieves high scalability by efficiently utilizing scalable interconnects with three key optimizations.

II. Background

A. Process of Neural Network

Neural network has two operational phases: *training* and *inference*. Training is a phase that tunes a neural network model, so that inference can use the trained model to conduct a given task. In this paper, we focus on training for simplicity, although our insight is generally applicable to the inference as well.

[1]The performance improvements depend on the aggregate throughput of neural network accelerators in a system. We use 256 neural network accelerators as a throughput target following [16].

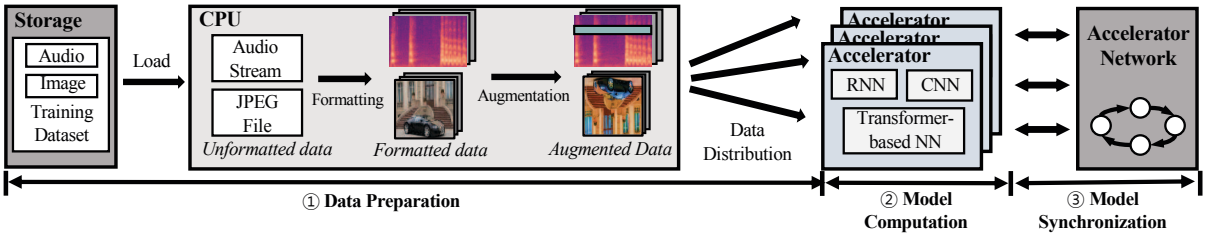


Fig. 4: Detailed process of neural network training.

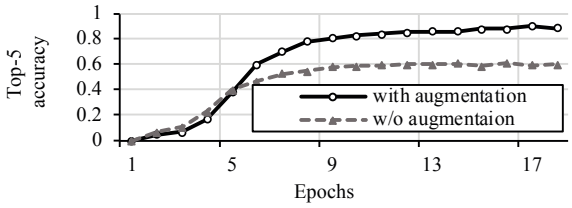


Fig. 5: Importance of data augmentation for higher model accuracy (Workload: Imagenet dataset on Resnet-50).

The process of neural network training typically consists of three steps: data preparation, model computation, and synchronization (Figure 4). The training repeats these three steps in series until the trained model reaches a certain condition (e.g., target accuracy). These processes are general regardless of input and/or neural network types.

Data preparation. Data preparation is the first and essential step of training, regardless of neural network models. It prepares input data with corresponding labels from a training dataset. We focus on image and audio datasets (e.g., Imagenet [9], Librispeech [34]) whose importance is well recognized. At first, a batch of data is loaded from the storage devices, and transformed into the forms specified by a neural network model (i.e., data formatting). Data formatting includes various compute-heavy operations such as decoding (e.g., JPEG→RGB format) and cropping to match the model-specific size (e.g., 256×256→224×224). For audio, we convert a stream of sound into a 'Mel spectrogram' [35], which is the STFT-based feature set of frames in the stream.

Another important role of data preparation is *data augmentation*, which generates synthetic data to enhance the model accuracy for unseen data [35], [37], [43]. For example, we can generate a horizontally flipped image, which does not exist in the original dataset. We can also add some noise into sound, so that the trained model performs well for noisy situations. *That is, by virtually increasing the size of the training dataset, we expect higher accuracy for the trained model.* We verify this claim in Figure 5; training with data augmentation shows 29.1% point higher accuracy than training without it. Sometimes, data augmentation cannot be strictly separated from the formatting; for example, we can augment data by selecting different crop basises during the formatting.

Model computation. In this step, we compute parameter adjustment factors for a neural network model using a processed data batch. First, a data batch from data preparation is fed to the model. Next, the model processes the data batch with learnable parameters (weight) using a series of layers, and produces prediction results. Then, the model computes loss (or error) by comparing the predicted result and the correct answer. Lastly, the adjustment factor (or gradients) for each parameter is calculated by back-propagating the error.

Model synchronization. The last step of training synchronizes the produced gradients from all accelerators. Because each accelerator gets a different batch, the models in the accelerators need to be synchronized by sharing the gradients in two steps. First, the gradients are aggregated from all accelerators. Then, we broadcast the aggregated gradients, so that each accelerator sees the global view of the model.

B. Acceleration of Neural Network Training

In this section, we briefly introduce some recent efforts to accelerate neural network training.

Data preparation. Data preparation has gained less attention compared to the other steps due to its 'relatively' low complexity. Rather than accelerating data preparation itself, the system designers use CPUs for computations of data preparation, and overlap data preparation with the other steps (e.g., Adam's pre-caching [8]). Such overlapping, which is next-batch prefetching, is easily achievable because the data preparation of the next batch does not depend on the results of the current batch. Therefore, the overhead of data preparation can be hidden from the overall latency, and data preparation has not been considered as the main acceleration target. However, we claim that data preparation becomes a critical performance bottleneck for up-coming neural network servers (Section III).

Model computation. Many hardware architects have designed specialized neural network accelerators [7], [23] to accelerate model computation. The main insight of such hardware acceleration is that the customized dataflow can efficiently exploit both the intrinsic data and computation patterns of model computation. Having a massive number of such dataflow also significantly increases performance by running the independent operations in parallel. Furthermore, accelerators using in-memory computing technologies (e.g., ReRAM [7]) have shown promising performance, area, and power benefits.

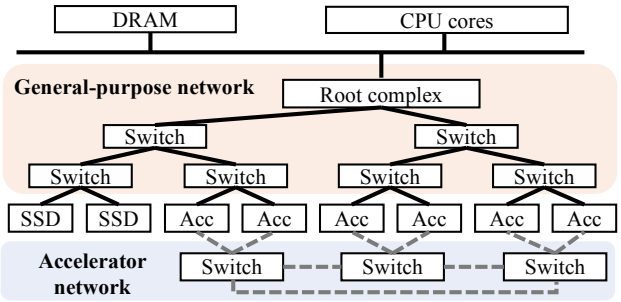


Fig. 6: Structure of a typical neural network server with Acc (neural network accelerator). Solid line: general-purpose link, dashed line: accelerator link.

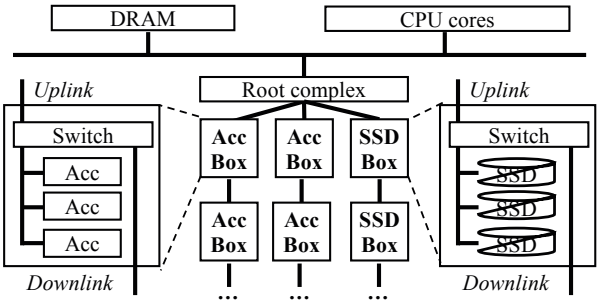


Fig. 7: Baseline neural network server architecture (omitted the accelerator interconnect, Acc=neural network accelerator).

Overall, these hardware accelerators successfully accelerate model computation and achieve orders of magnitude higher efficiency (i.e., performance/watt/area) over traditional hardware (e.g., CPUs). The rapid improvement has been seen among generations: more than 10,000× in Figure 2a.

Model synchronization. Lastly, many efforts are made to reduce the model synchronization overhead for a large number of neural network accelerators. We summarize such efforts in three folds. First, the interconnect designers have proposed new communication channels (e.g., NVLink [33]) dedicated to neural network accelerators. By sacrificing some degree of generality, they can have simple network stacks and significantly boost the inter-device communication bandwidth.

Second, the system designers exploit intrinsic communication patterns in training. For example, NVIDIA's NCCL [30] uses communication primitives like ring-/tree-based aggregation, which shows scalable synchronization performance. These ideas exploit all the available interconnect bandwidth and the computation capability in a system to guarantee scalable synchronization. As Figure 2b shows, the model synchronization latency saturates at the double of the latency with two accelerators. Therefore, the larger number of accelerators does not imply higher model synchronization overheads. Other researchers also report high scalability of the synchronization in real/simulation environments [25], [26] (e.g., NVIDIA's test with 24K GPUs [20]).

Third, the algorithm designers have found a way to increase the batch size without loss of model accuracy. Traditionally, the large batch size is known to lead to lower accuracy due to the training instability. However, recent efforts [13] prove that using a proper learning rate can remove such instability. As the model synchronization latency does not depend on the batch size (i.e., each neural network accelerator only sends the locally aggregated gradients of the parameters), the model synchronization overhead can be relatively reduced with a large batch (compared to the model computation overhead).

As a result, neural network training severs start to have many accelerators and achieve significant improvement in system throughput.

C. Typical Neural Network Server Architecture

A neural network server has CPU cores, host memory, neural network accelerators, data-source devices, and interconnect networks among them (Figure 6). The server uses CPUs to manage underlying devices and host memory to temporally buffer the data during the communication among the devices. The neural network accelerators execute compute-heavy neural network applications (e.g., Transformer [46]) using datasets (e.g., Librispeech [34]) provided by data-source devices (e.g., SSD). Because of the huge computation demands of the neural network applications, it is common to deploy multiple accelerators in a node (e.g., 16 GPUs in DGX-2 [33]).

The server isolates communication channels among devices into two interconnect networks. The first network is a *general-purpose interconnect* (i.e., PCIe) which connects all devices. In PCIe, devices and switches form a tree structure; The root complex (RC) is the root of the tree, PCIe-switches become internal nodes, and the devices are located at the leaves. The other network is an *accelerator interconnect* (e.g., NVLink [33]) which only connects neural network accelerators in the server.

The main difference between the two interconnects is in their bandwidth. The general-purpose interconnect prioritizes the compatibility for commodity devices over throughput, so their performance improvement is much slower than the accelerator interconnect. In contrast, the accelerator interconnect shows huge performance improvement thanks to its highly customized network stacks. For example, the accelerator interconnect in DGX-2 [33] provides 9.4× higher throughput than the general-purpose interconnect.

III. Motivation

In this section, we describe a baseline large-scale server architecture and its limitations to introduce design guidelines of large-scale neural network servers and motivate TrainBox.

A. Baseline Large-scale Server Architecture

Building large-scale training systems becomes practical thanks to the advance in accelerators and communication methodologies. Figure 7 shows an example naive large-scale server with such advancements. We carefully design the

TABLE I: Summary of workloads.

NN Type	Name	Task	Batch size	Model size (MB)	Throughput (Sample/s)
CNN	VGG-19	Image classification	2,048	548.0	3,062
	Resnet-50		8,192	97.5	7,431
	Inception-v4		2,048	162.7	1,669
RNN	RNN-S	Image captioning	4,096	1.0	12,022
	RNN-L		2,048	16.0	6,495
Transformer (TF)	TF-SR	Speech recognition	512	268.3	2,001
	TF-AA	Audio analysis	512	162.5	2,889

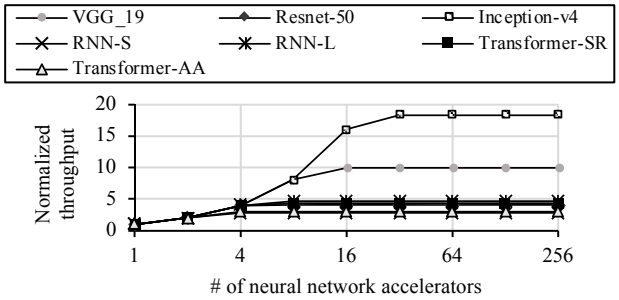


Fig. 8: Scalability of various neural network models. For each model, we normalize the throughput values to the throughput value with a single neural network accelerator.

baseline to reflect the structure of modern training servers. Especially, we rely on the difference between NVIDIA's DGX-1 and DGX-2, the two most popular training systems at the time. For simplicity, we omit the accelerator interconnect and use a notion of 'box'; a box consists of multiple devices and several PCIe switches, and has two external ports (an uplink and a downlink). To scale the number of devices, we chain the boxes from the root complex by connecting the uplink and the downlink of two boxes (details in Section V-D).

This server achieves scalability *in terms of the number of accelerators* in two folds. First, we put all devices in a node on a PCIe interconnect, which is designed with a scalable tree topology. Second, it relies on a ring-like topology [16], [25], [33] for the accelerator interconnect, which is easily formed with interconnect switches (e.g., NVSwitch [33]).

This 'scale-up' approach has several benefits over 'scale-out' approaches, which connect separate nodes to scale.[2] First, we can reduce total cost of ownership (TCO) for host resources; scale-up can amortize host resources while scale-out requires dedicated resources for each node [18] (e.g., one node with 256 accelerators vs. 256 nodes with one accelerator per node). Second, scale-up is much easier to deploy high-speed inter-accelerator networks (e.g., 300 GB/s NVLink). In contrast, it is difficult to achieve such high speed for the inter-node communication due to complex network stacks (e.g., 100 Gbps NIC). This leads to the synchronization bottleneck in scale-out systems (e.g., a scale-out system with 96 DGX-2 shows only 39.7× improvement over one DGX-2 in MLPerf results [31]). Lastly, we can use a simple single-node infrastructure (e.g., Caffe [22]) because neural network accelerators in scale-up can be managed by a single OS; users also can easily train their models beyond a capability of a single node without considering the inter-node communication. These claims rationalize the recent trends of neural network servers (e.g., 8 GPUs of DGX-1→16 GPUs of DGX-2).

[2]This paper focuses on the case when the server is dedicated to a single training job although our discussion is applicable for multi-job training [14]; the multi-job (or multi-modal) training shows lower synchronization overhead because of a smaller number of accelerators used for each job. Scale-up can still achieve lower TCO for this case thanks to the reduced host-side resources.

B. Limitations of the Baseline Architecture

We then profile the performance and scalability of the baseline to identify the bottlenecks in large-scale servers.

1) Experimental setup: We set *the throughput of 256 TPU v3-8 instance* as a target goal, which is the largest amount of neural network accelerators in the literature [16]. Our analysis is applicable to any neural network accelerators with high throughput; we use TPU v3-8 as it is publicly available. As the scale of 256 accelerators is smaller than NVIDIA's synchronization test with 24K GPUs [20], it also guarantees scalable model synchronization.

For workloads, we use seven popular neural network models covering three types (Table I). The first three models are convolutional neural networks (CNNs) for image classification. The next two models, RNN-S(mall) and RNN-L(arge), are LSTM-based recurrent neural network (RNN) models for image captioning [48]. The last two models are Transformer for speech recognition (SR) and audio analysis (AA). We set the batch size as the largest batch size that a single TPU v3-8 instance can run. For datasets, we use Imagenet [9] (stored in 256×256 JPEG) for image tasks, and Librispeech [34] (stored in sound streams of 6.96s on average) for audio/speech tasks.

We construct and profile the baseline architecture using our machine having two-socket Xeon CPUs (i.e., 48 physical cores). For framework, we optimize Caffe with the state-of-the-art data preparation library (NVIDIA DALI [32]). We also apply well-known software optimization techniques such as batching, software pipelining, and data partitioning for less lock contention to the baseline.

We use *virtual devices* to emulate a large-scale system. Each thread running a virtual device sleeps predefined time (=computation time of TPUs + synchronization time in Figure 2b) to mimic the model computation and synchronization overheads, and does not perform any computations to minimize the profiling overhead. However, this approach cannot include some overheads such as data transfer and SSD read. Therefore, we separately measure their overheads and adjust the virtual-device results with them. In this way, we can accurately model a system with many accelerators; we verify the accuracy of this methodology within our system setup.

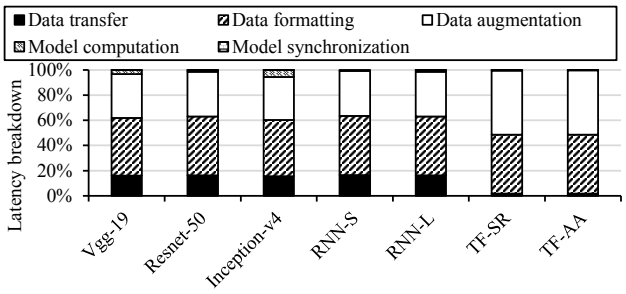


Fig. 9: Latency decomposition of neural network models.

2) Result – Limited scalability: We first test the throughput scalability of the baseline (Figure 8). As the number of accelerators increases, the throughput is expected to increase. However, we observe that the system throughput saturates at the very early stage; after 18 neural network accelerators, all models do not benefit from more accelerators.

Our analysis reveals that the baseline struggles from the *high data preparation overhead* (Figure 9). Unlike a common belief that model computation and synchronization would be critical, the baseline with numerous accelerators suffers from extremely high overhead from data preparation. *On average, data preparation accounts for 98.1% of the total latency when processing a batch.* As such, even with next-batch prefetch, the data preparation determines the system throughput and we cannot expect practical gains despite many accelerators.

C. Bottleneck Analysis

To identify the cause of the high data preparation overhead, we profile the resource utilization of three host resources.

CPU. Our profiling result shows that the baseline requires an infeasible number of CPU cores not to make the data preparation a major bottleneck (Figure 10a). To achieve the target throughput (i.e., 256 accelerators), the system should support up to 4,833 cores, which is 100.7× more CPU cores than DGX-2. In contrast to DGX-2 with V100, whose accelerator:CPU-core ratio is 3:1, high-performance accelerators and innovations on the model synchronization lead to a higher ratio of 18.9:1. As typical nodes support two or four CPU sockets, the amount of required cores is infeasible.

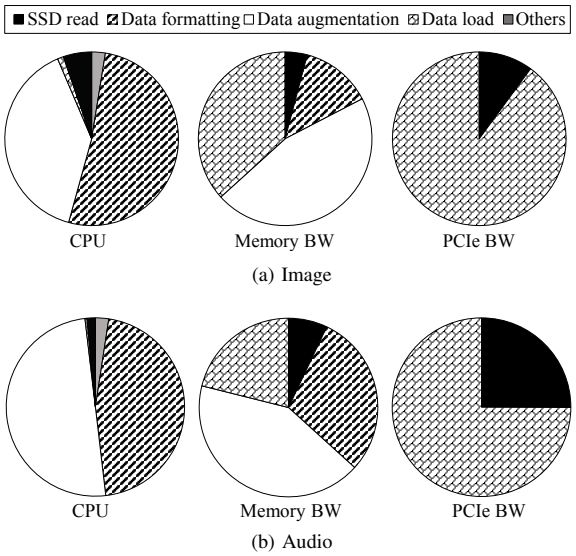


Fig. 11: Decomposition of the host resource consumption.

Moreover, it is easy to expect higher ratios with upcoming neural network accelerators and their interconnects.

To identify the reason for such high overhead, we check CPU cycles for each operation (CPU in Figure 11a and 11b). For both input types, data preparation takes most of the CPU resources. Especially, data formatting and augmentation are the heavy contributors of such CPU consumption, which indicates a dire need of acceleration.

Memory. We also observe that the baseline performance is limited by memory bandwidth (Figure 10b). With 256 accelerators, the baseline demands up to 17.9× higher memory bandwidth than what DGX-2 provides (i.e., 239 GB/s). We decompose memory bandwidth utilization to identify the cause of the overhead (Memory BW in Figure 11a and 11b). Overall, both types of input consume much memory bandwidth for data preparation. Data load for image (audio) from the host memory to the neural network accelerators takes 36.7% (21,1%) of memory bandwidth along with 59.2% (71.9%) from data formatting and augmentation. The larger consumption of the data load over the SSD read stems from amplified data size due to decompression, type casting (char→ float), and SFFT

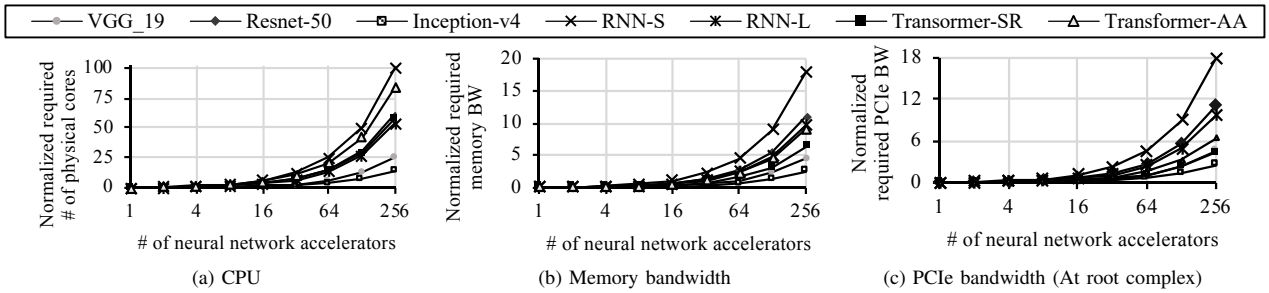


Fig. 10: Required host resources normalized to the resource of DGX-2.

for Mel spectrogram in data preparation.

Interconnect. Lastly, we identify that the available PCIe bandwidth also limits the throughput of the baseline (Figure 10c). With 256 accelerators, the server should support up to 18.0× higher PCIe bandwidth than DGX-2. Our analysis shows that high PCIe requirement also comes from data preparation. (Figure 11). As the computation of data preparation does not involve any devices, the data copy (SSD→memory and memory→accelerators) dominates the PCIe bandwidth consumption, and makes the bottleneck at the root complex.

D. Limitations of Naive Solutions

One possible solution to reduce the computation overhead of data preparation is skipping data augmentation. For some cases (e.g., to check the effect of neural network's structural changes without touching other hyperparameters), we may use a fixed dataset and save the processed data to storage in advance to reduce the run-time overhead. However, the fundamental goal of training is to have the 'best' accuracy (rather than inter-model comparisons) and the augmentation is one of the important hyperparameters [37] (Figure 5). Therefore, we should make our system perform the data preparation.

The other candidate is saving the augmented data in the storage devices in advance (i.e., static data preparation). However, this approach is impractical due to the excessive storage overhead caused by heavy randomization in data preparation. As an example, we provide the storage requirements of random cropping which converts a 256×256 RGB image to 32×32 different images of 224×224 size. Considering a 224×224 RGB image takes 0.15 MB and the Imagenet dataset has 14 million images, static data preparation requires about 2.2 PBs (32×32×0.15MB ×14M). Moreover, the storage overhead further increases with other augmentation techniques (e.g., random noise addition) and larger datasets. As such, this leads to huge storage overheads in terms of bandwidth and capacity. Therefore, we need to perform on-line data preparation for better storage efficiency.

E. Design Guidelines

Based on the analysis of the neural network server at scale, we set four design guidelines for large-scale server designs.

- **Throughput scalability through scale-up.** The system throughput should increase as the number of accelerators increases in a scale-up manner.
- **Data preparation acceleration.** We should accelerate data preparation because of the bottleneck shift caused by high-performance neural network accelerators and new ideas for efficient model synchronization.
- **Commodity-hardware-based acceleration.** The acceleration should minimize the use of the host resources to guarantee high scalability. Also, we should use commodity hardware devices and general-purpose interconnects for low accelerator design costs.
- **On-line data preparation and large coverage.** We should run data preparation on-the-fly for high accuracy and storage efficiency. The server should be effective for various neural network models for diverse use cases.

IV. TrainBox: Extreme-scale Neural Network Server Architecture

In this section, we introduce TrainBox that enables scalable neural network acceleration with high-performance neural network accelerators and the low synchronization overhead.

A. TrainBox Overview

TrainBox is a neural network server architecture for a large number of neural network accelerators and high-speed interconnects. To follow the design guidelines, TrainBox relies on only commodity hardware (i.e., FPGAs, PCIe, Ethernet) for high scalability and generality in a single node. It also does not require any changes on neural network accelerators, their interconnects, and neural network models.

TrainBox extends the baseline server by applying three optimizations in series. First, TrainBox accelerates CPU-heavy operations of data preparation by offloading them from host CPUs to scalable, commodity hardware accelerators. Second, TrainBox enables the peer-to-peer communication to mitigate the excessive pressure on the host memory. Lastly, TrainBox clusters the devices based on the communication patterns to reduce PCIe bandwidth consumption and deploys a dedicated interconnect network among the clusters to adapt to the workload variability. As a result, TrainBox effectively reduces pressure on host-side resources and achieves workload adaptability.

In the following sections, we describe each optimization.

B. Step 1: Acceleration of Data Preparation

We start from the baseline in Figure 12, which requires excessive CPU, DRAM bandwidth, and PCIe bandwidth. As the first optimization, we offload CPU-heavy operations of data preparation to a scalable array of hardware accelerators (Figure 13). Although the data preparation throughput should increase with more accelerators, we cannot add CPU chips on a single server due to practical reasons (e.g,. form factor, cache coherency). Therefore, we decide to use PCIe-attached accelerators, which can be scaled easily with PCIe switches. We offload data formatting and augmentation, which show high CPU utilization (Figure 11).

To enable acceleration, we put another box called preparation (prep) boxes under the PCIe tree. The structure of a data preparation box is the same as other boxes except that it consists of the data preparation accelerators instead of neural network accelerators. Each data preparation accelerator processes the data from SSDs with the help of the host memory (as a temporal buffer). After a data preparation accelerator finishes the computation, it requests the host CPUs to transfer the processed data to the neural network accelerators.

C. Step 2: Peer-to-peer Communication

Our next optimization target is memory bandwidth. We notice that most of the memory bandwidth overhead comes from

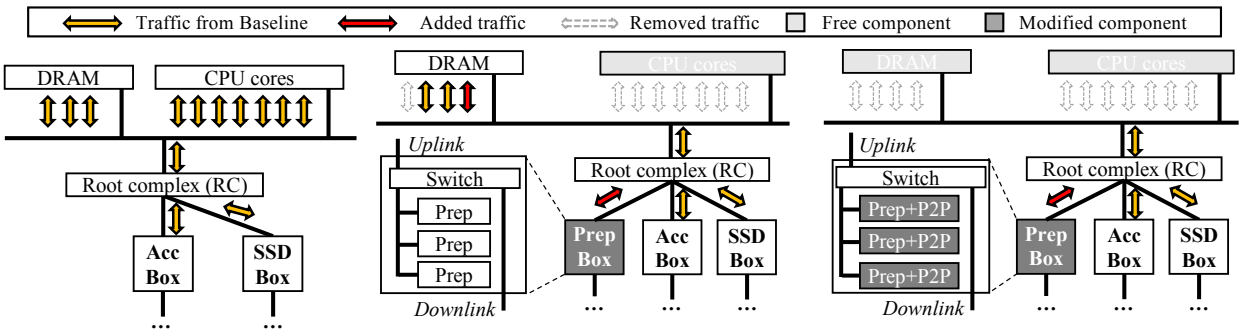


Fig. 12: Baseline.

Fig. 13: Step #1. Computation acceleration using hardware accelerators.

Fig. 14: Step #2. Peer-to-peer (P2P) communication for intermediate data.

simple buffering and CPU-side memory accesses (Figure 11). After offloading computation to the hardware accelerators, the latter is removed, but we see the additional buffering overhead for data preparation accelerators.

Our second optimization removes the memory pressure from data preparation by allowing the peer-to-peer (P2P) communication among devices (Figure 14). Because the host now simply transfers the input data without further processing, we can safely adopt the P2P communication. Since the P2P communication does not require any host memory, it can free host memory from the communication.

To enable the direct transfer, we rely on the PCIe's P2P feature [36]. At the boot time, the system assigns a unique PCIe address ranges to each PCIe device and port of PCIe switches. Later, PCIe switches forward (rather than broadcast) packages based on their destination address and the address range of each port (connected to either an end device or another switch), which are stored in the switch's internal registers. PCIe switches are designed to support full inter-port bandwidth, so the system exploiting P2P can efficiently bypass the host memory access for the data communication.

In data preparation accelerators, we put a P2P handler which manages the P2P communication for SSDs→data preparation accelerators→neural network accelerators. As a result, we can free unscalable host memory from data preparation and can rely on huge aggregated memory bandwidth from scalable PCIe devices.

D. Step 3: Communication-aware Clustering

As the previous two optimizations address the CPU and memory bottlenecks, the remaining one is the PCIe bandwidth. Unfortunately, the pressure on the PCIe RC becomes double over the baseline because the datapath is now SSDs→**RC**→data preparation accelerators→**RC**→neural network accelerators (vs. SSDs→**RC**→neural network accelerators).

We notice that the data communication pattern in data preparation is simple; a single data batch is consumed by a single neural network accelerator. In addition, because a PCIe switch forwards PCIe packets only to the destination, we can minimize the datapath between devices. That is, if we *localize* the communication within a PCIe switch, the upper levels of the PCIe tree do not suffer from high pressure.

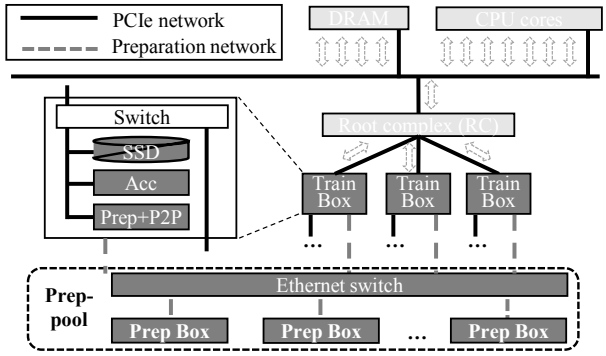


Fig. 15: Step #3. Device clustering and the prep-pool for low PCIe bandwidth consumption.

Therefore, TrainBox clusters the devices based on the communication patterns (Figure 15). Rather than aggregating devices based on the type, we place the devices in the same box based on *the datapath*. TrainBox deploys a group of cluster boxes (train boxes) consisting of SSDs, neural network accelerators, and data preparation accelerators. Within a box, data preparation accelerators control local data movements among devices; the host-side software partitions training data and configures preparation accelerators accordingly (details in Section V-A). As a result, all data are consumed within the box and upper-level links do not suffer from the PCIe packet explosion.

One drawback of this approach is that the number of data preparation accelerators (=data preparation throughput per box) is statically determined at the deployment, which makes it difficult to meet the resource requirements for diverse neural network models (=various throughput requirement). Furthermore, the required resource also depends on the original data features (e.g., sampling rate, image size). As a result, it is difficult for a train box to meet the various requirement without changing the physical configuration.

To address this problem, TrainBox deploys a data preparation network through Ethernet interconnects. We choose to

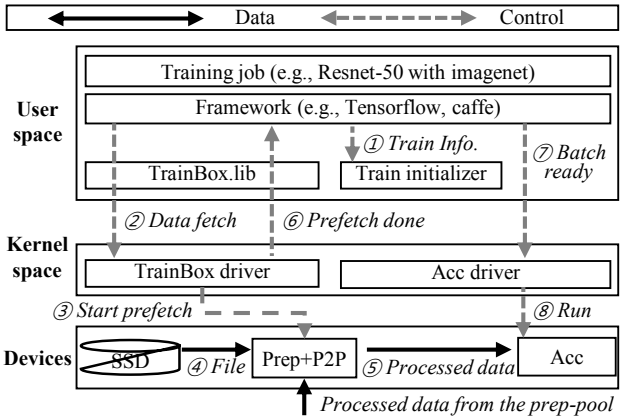


Fig. 16: System overview of TrainBox showing the hardware devices, the kernel space, and the user space.

dedicate network rather than sharing existing PCIe not to incur contentions on the PCIe. In addition, we use Ethernet because it is standard and provides comparable throughput over PCIe (100Gbs=12.5GB/s vs. 16GB/s). When the required data preparation resource is low to meet the required throughput, the data preparation accelerator requests further processing to the data preparation boxes in the data preparation network (prep-pool). To enable the data preparation network, each preparation accelerator in a train box has an Ethernet link connected to the data preparation network.

V. Implementation Details

A. System Overview

To enable TrainBox, we modify the overall system including the user space, the kernel space, and devices (Figure 16). In the user space, we mainly add two modules: a train initializer for data distribution and TrainBox.lib for data preparation. Before the actual training process starts, ① the train initializer distributes the data to SSDs in each train box and assigns additional resources from the prep-pool. To do so, the initializer gets training information (e.g., network model, input size, batch size) about a to-be-run training job. Next, the initializer feeds random dummy batches to a neural network accelerator to measure the per-batch execution time with the help of the framework. It then calculates the required throughput of data preparation using the measured execution time and the synchronization model.

If the required throughput is less than the capability of data preparation accelerators in a train box, the initialization finishes. Otherwise, the train initializer requests extra resources from the prep-pool. It first calculates the number of required data preparation accelerators by dividing throughput by per-accelerator throughput (measured offline). Then, the train initializer accesses a global resource manager/scheduler (e.g., Mesos [15]) to allocate extra accelerators. Finally, the train initializer assigns the allocated accelerators from the prep-pool and provides the information of them (e.g., Ethernet address) to groups of data preparation accelerators in train boxes; the preparation accelerators in the same group (within a train box) share additional accelerators.

After data distribution, the framework is ready to run training. As the first step, ② the framework lets TrainBox.lib prefetch the data to neural network accelerators. Upon receiving a prefetch request, the library communicates with the TrainBox driver, ③ so that the data preparation can start in the data preparation accelerators. Then, ④ the preparation accelerator fetches the training data from SSDs within the same train box, ⑤ performs the data formatting and augmentation, and ⑥ loads the processed batch to neural network accelerators in the same train box. If extra data preparation throughput is required, the data preparation accelerator offloads some computation to pre-allocated accelerators in the prep-pool, whose information is provided by the train initializer at the train initialization phase. Finally, ⑦ the data preparation accelerator delivers the batch ready signal to the neural network accelerator driver through the framework, and ⑧ the neural network accelerators get a run signal.

Note that TrainBox also lowers user/kernel mode switching overhead over the baseline. While the baseline incurs many user/kernel switching due to frequent interaction between applications and device drivers (e.g., NVMe), our TrainBox offloads such interactions to the accelerators and thus requires less interaction between applications and drivers.

B. Which Device for Data Preparation?

Although TrainBox does not limit the device type for the data preparation acceleration, we implement TrainBox with FPGAs due to the following reasons. First, because of the high diversity of training datasets (e.g., audio, compression format), the data preparation accelerator needs to have enough flexibility. Therefore, we mainly care about Xeon Phi, GPUs, and FPGAs, which are programmable, rather than integrating data preparation functionalities in neural network accelerators. This also matches our design guideline of using commodity devices and does not increase their design complexity.

We mainly use FPGAs because of its higher throughput for data preparation. In contrast, PCIe-attached Xeon-phi has limited throughput due to its simpler microarchitecture and lower clock frequency (2×) than our Xeon CPUs [19]. Even only considering frequency difference leads to much higher core/accelerator requirements (more than 37.8 cores/accelerator or 0.52 device/accelerator). We also observe that GPUs cannot efficiently handle some operations, especially data formatting, due to their irregularity. For example, there is no good parallel algorithm for the Huffman decoding phase in JPEG decoding [40]. As a result, even NVIDIA's DALI heavily relies on CPUs, which cannot avoid the performance bottlenecks of the baseline. For audio, generating Mel spectrograms needs many small FFT computations, for which FPGAs show higher performance than GPUs [39].

In addition, as data preparation is conducted on the general-purpose PCIe interconnect, the data preparation accelerator should be able to manage direct P2P communication for any

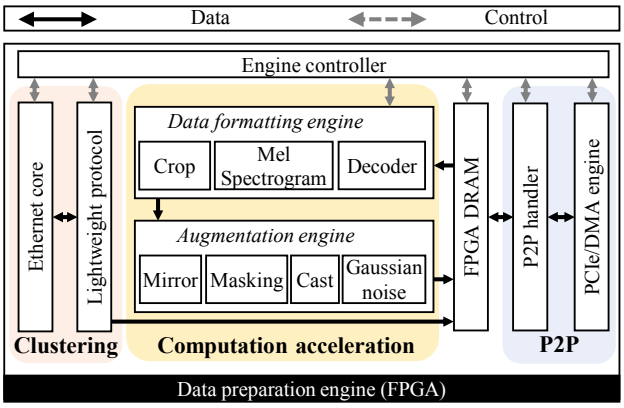


Fig. 17: Microarchitecture of FPGA-based data preparation accelerators to enable three optimizations.

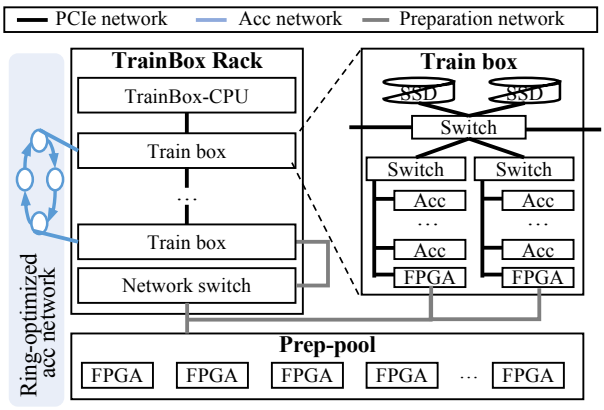


Fig. 18: Overview of rack-scale TrainBox. Rack-scale TrainBox connects multiple Train boxes through Ethernet switch and considers the physical limitation of PCIe switches.

devices. However, for GPUs, such functionality is limited to the selected device pairs. Recent FPGAs are also equipped with a high-speed network interface (e.g., 100 Gbps [51]), which easily enables the prep-pool.

C. Microarchitecture of FPGA Accelerators

Figure 17 shows the microarchitecture of our data preparation accelerator on an FPGA. A data preparation accelerator consists of a TrainBox controller for the device management within a train box, a clustering module to communicate with the prep-pool, a computation acceleration module for the computation acceleration[3], on-board DRAM for data buffer, and a P2P module for direct inter-device communication.

Upon receiving a prefetch request, the TrainBox controller lets the P2P module fetch the data from SSDs to the on-board DRAM in the FPGA. Then, the data enter the data transformation pipeline (from a data formatting engine to an augmentation engine) in the computation module. If computation throughput is lower than the required bandwidth (determined by the training initializer in advance), the TrainBox controller uses the clustering module to use data preparation accelerators in the prep-pool. Finally, the P2P module transfers the complete batch to the specified neural network accelerators.

To enable P2P communication between an SSD and an FPGA, we modify DCS-engine [1], [24], and use it as the P2P handler. DCS-engine is an FPGA-based peer-to-peer communication mediator, which supports standard PCIe devices including NVMe SSDs. Specifically, we implement NVMe command generators, and place NVMe command and completion queues in the FPGA memory. In this way, FPGAs can issue NVMe commands and fetch the data from SSDs. For P2P with an accelerator, we assume pinned memory communication like GPUs. As a result of P2P communication, we do not use any host-side resources for SSD accesses.

[3]Current implementation does not cover some preparation operations (e.g., shuffling, weighted sampling) which have dependency among items. TrainBox can support them in either data replication among SSDs or communication through the prep-pool network.

We implement the data preparation accelerators using Verilog and intellectual properties (IPs) from Xilinx on Xilinx Vivado 19.01. Specifically, we heavily optimize data preparation (e.g., double buffering) in Verilog since we expect their high performance impact. For the P2P communication, we mainly use Xilinx PCIe/DMA engine and attach our control logic. We deliver the data preparation functions on FPGAs to the end-users as libraries not to increase the programming burden. When a user wants to add a new data preparation functionality (e.g., new input form such as video), they need to implement it through RTL or high-level synthesis (HLS).[4] Then, we can program FPGAs using techniques such as partial re-configuration [49]; most of the interfacing logics remain unchanged, and only the computation acceleration part of the accelerator is changed.

D. Rack-scale TrainBox

Now, we introduce how to realize the TrainBox at rack scale. Figure 18 shows rack-scale TrainBox with an external prep-pool. In a TrainBox rack, we place TrainBox-CPU with multiple instances of train boxes. Train boxes are connected in series, starting from the TrainBox-CPU, which is the host for TrainBox. We assume a ring-optimized accelerator network for neural network accelerators in train boxes following DGX-2 [33] or TPU pod's style [12].

We assume a train box as a PCIe expansion system, which has several PCIe switches (e.g., [42]). This box-based system enables easy deployment as each expansion system has dedicated power supply units and is isolated in terms of physical constraints. Following popular DGX-2 [33] and Supermicro [41] styles, we assume eight neural network accelerators in a train box. Since high-end PCIe switches (e.g., PEX8796 [44]) has up to six links (five for downlinks and one for an uplink), we place four neural network accelerators

[4]We plan to add easier programming support as a future work. We believe HLS is promising for this purpose considering their rapid evolution and programmability [50].

TABLE II: Resource utilization on an FPGA (Image version).

		LUTs	FF	BRAM	DSP
Preprocessing Engine	Jpeg decoder	704K (59.6%)	665K (28%)	0	1040 (15.2%)
	Crop	0.5K	0.3K	0	27 (0.4%)
	Mirror	6.5K (0.5%)	4.7K (0.2%)	0	381 (5.6%)
	Gaussian noise	24.5K (2.1%)	33K (1.4%)	80 (3.7%)	400 (5.8%)
	Cast	5.7K (0.5%)	3K (0.1%)	0	240 (3.5%)
Ethernet + Protocol parser		166K (14%)	169K (7.1%)	1024 (47.4%)	0
P2P Handler		22.7K (1.9%)	24.7K (1.1%)	153 (0.4%)	0
Total		78.7%	38.1%	51.5%	30.5%

TABLE III: Resource utilization on an FPGA (Audio version).

		LUTs	FF	BRAM	DSP
Preprocessing Engine	Spectrogram	622K (52.6%)	755K (31.9%)	228 (10.6%)	0
	Masking	21K (1.8%)	17K (0.7%)	53 (2.4%)	260 (3.8%)
	Norm	14K (1.1%)	11K (0.5%)	0	0
	Mel Filter bank	103K (8.7%)	119K (5.0%)	208 (9.6%)	572 (8.4%)
Ethernet + Protocol parser		166K (14%)	169K (7.1%)	1024 (47.4%)	0
P2P Handler		22.7K (1.9%)	24.7K (1.1%)	153 (0.4%)	0
Total		80.2%	46.3%	77.1%	12.2%

and an FPGA under a PCIe switch and connect two of such switches using another switch having two NVMe SSDs.

For the channel to the prep-pool, we use top-of-rack Ethernet switches, whose practicality is verified by in the real environments (e.g., Catapult [3]). We use Ethernet to connect FPGAs rather than PCIe to reduce the contention of the PCIe tree. Because the Ethernet links on the latest FPGAs support comparable bandwidth over PCIe (e.g., dual 100 Gbps), we can utilize Ethernet with next-batch prefetching (i.e., no latency limitations). In contrast, using the PCIe link to serve both inter-device communication and FPGA networks makes it the bottleneck because each end device has only one PCIe link.

We can realize the prep-pool in multiple ways. First, we can deploy disaggregated FPGA racks (or FPGA-boxes under TrainBox Rack) which only consist of FPGAs and are shared by multiple jobs. Second, recent techniques (e.g., MSR's Catapult [3]) provide FPGA pools by abstracting distributed FPGAs in a datacenter. Lastly, if a single TrainBox rack serves multiple jobs or some train boxes are unused, we can leverage FPGAs in underutilized train boxes as a prep-pool. Because each workload demands the different amount of resources (Figure 10), there can be underutilized FPGAs that can be used by overutilized train boxes.

VI. Evaluation

A. Experimental Setup & Methodology

We prototyped TrainBox on a real machine and evaluated it using the workloads in Table I. We built TrainBox using three FPGAs (Xilinx XCVU9P): one for data preparation acceleration, another for the prep-pool, and the other for ASIC emulation. Then, we extended Caffe to support TrainBox for the data preparation of JPEG-image and audio files using the FPGAs. For the baseline, we use the same methodology in Section III (i.e., 48 physical cores in DGX-2).

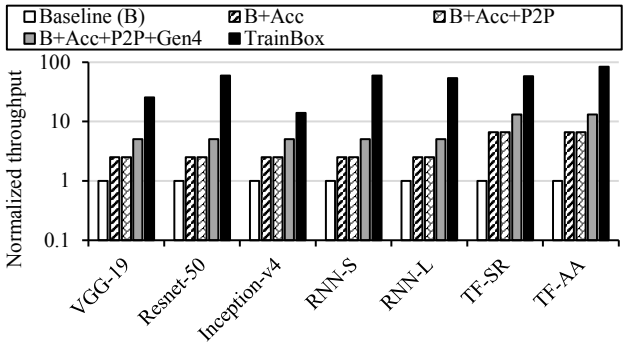


Fig. 19: Impact of TrainBox's optimizations with 256 neural network accelerators (ACC: computation acceleration, P2P: peer-to-peer communication, TrainBox: B+Acc+P2P+Clustering).

As the construction of rack-scale TrainBox is impractical at a research level due to its large scale and unavailability of TPUs, we built a performance model of TrainBox by profiling the prototype (mainly, CPU, memory, and PCIe overhead). Then, we implemented a system-level simulator using the model, adjusted it with the real numbers from our own measurements, and verified data from prior studies. Specifically, to get the model computation overhead, we measured the throughput of TPU v3-8 on Google Cloud for the various models and configurations. For the model synchronization overhead at large scale, we carefully built a performance model based on the ring communication and assumed an NVLink-like interface based on [20], [25].

Our simulator is accurate because of two reasons. First, we built our model using real measurements on the prototype (rather than pure simulations), and validated its accuracy with the various configurations. Second, as training is throughput oriented, the impact of latency variations on the overall throughput is small [25] thanks to pipelining/next-batch prefetching.

B. FPGA Resource Utilization

Table II and III show the resource utilization of a data preparation accelerator for the image and audio types, respectively. For image, we implemented basic data preparation operations; among them, the JPEG decoder takes most of the resources due to its high complexity. For audio types, we add masking and normalization. Overall, our FPGA-based data preparation engine consumes, on average, 79.5%, 42.2%, 64.3%, and 21.4% of LUTs, FF, BRAM, and DSP resources, respectively. We emphasize that these are the example configurations and we can adjust portion and/or functionality based on user's demand.

C. Throughput

Now, we evaluate the throughput of TrainBox (Figure 19). Our TrainBox shows, on average, 44.4× higher throughput with 256 neural network accelerators than the CPU-based

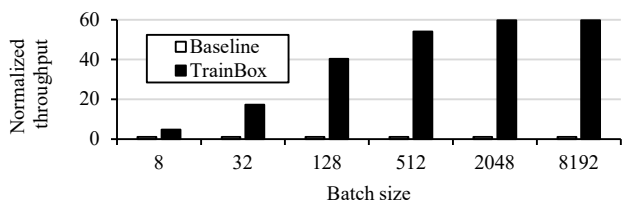


Fig. 20: TrainBox's effectiveness for various batch size with 256 accelerators (Resnet-50).

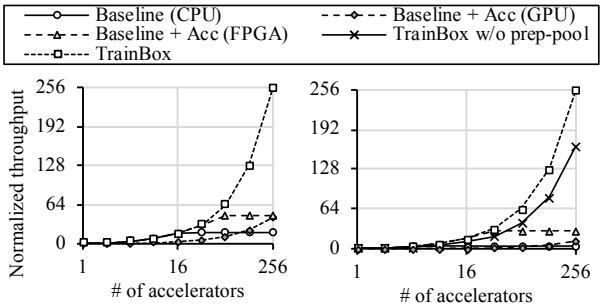


Fig. 21: Scalability test for Inception-v4 and TF-SR ('Baseline+Acc+P2P' = 'Baseline+Acc'). For Inception-v4, TrainBox without prep-pool is not shown because its performance is same as TrainBox.

baseline. The performance improvement is higher for neural network models with larger throughput due to the heavier pressure on the data preparation; the improvement (84.3×) is the largest with TF-AA, due to its heavy resource requirement and the success of FPGA-based acceleration.

We also confirmed that each proposed optimization successfully boosts the throughput. First, the acceleration of the computations in data preparation boosts the throughput 3.32× on average. The acceleration benefit for the audio types is slightly larger due to audio types's higher computation loads for data preparation. However, the P2P communication does not increase the system throughput since the acceleration increases the PCIe overhead (Section IV-D). With clustering, we get another performance improvement of 13.4× on average. At this point, both memory and PCIe overhead are resolved, and the improvement of the P2P communication is now visible. While doubling the PCIe bandwidth (B+Acc+P2P+Gen4) is beneficial, TrainBox without Gen4 shows even higher improvement, indicating that the bottleneck stems from the inefficient datapath.

We also test TrainBox's superiority over the baseline for the various batch sizes. In this experiment, we use Resnet-50 because it has the largest batch among our workloads. Figure 20 shows that TrainBox improves the performance over the baseline for every batch size. The speed-up of TrainBox is higher for larger batches due to relatively reduced inter-communication overhead and better efficiency of neural network accelerators (i.e., higher resource utilization with a larger batch).

D. Scalability

Next, we show that TrainBox achieves good performance at various scale (Figure 21). For clarity, we only show Inception-v4 and TF-SR as examples. The performance of the CPU baseline saturates at 18.3 and 4.4 neural network accelerators due to the limited number of CPU cores for Inception-v4 and Transformer-SR, respectively. At small scale, data preparation acceleration using GPUs shows lower throughput than the baseline due to the lack of GPU resources (1:4 ratio). Only when the number of GPUs is large enough, its throughput becomes higher. In contrast, FPGA-based data preparation acceleration quickly shows high throughput. This proves that FPGAs are a better match for data preparation acceleration. At larger scale, while the performance of Baseline+Acc saturates, removing the data movements of memory and PCIe channels boosts the throughput significantly.

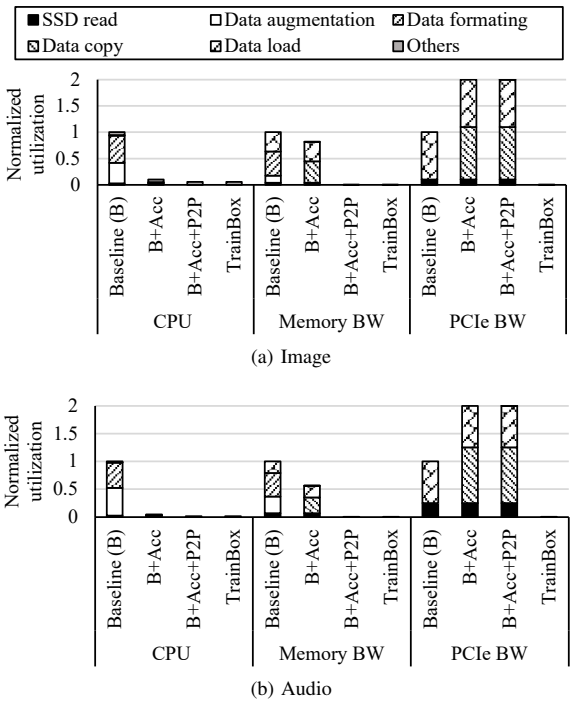
In addition, the graphs show the importance of the prep-pool for workload adaptability. While Inception-v4 reaches the target throughput without the prep-pool, Transformer-SR without the prep-pool shows the little performance loss due to the limited FPGA resources for data preparation. Since, the audio preparation requires much higher computation capability than images, the system deployed for images may suffer from the little performance loss like this. In this case, the prep-pool provides the additional performance improvement with 54% more FPGA resources from the prep-pool; the system reaches the target throughput without changing the server structure.

E. Host Resource Utilization

Now, we investigate the effectiveness of the proposed optimizations for the host resource utilization. Overall, TrainBox with our three optimizations significantly reduces the host resource utilization (Figure 22), so that the performance can scale regardless of the host resources. For the CPU utilization, the computation acceleration of the data preparation significantly reduces the CPU resource consumption. The P2P communication further reduces the CPU utilization by removing the NVMe driver overhead. For the host memory bandwidth consumption, the P2P communication removes most of the memory bandwidth consumption. The computation acceleration also reduces the memory pressure as the host does not need to access the data in the memory for the computation. Lastly, the computation acceleration increases the PCIe bandwidth due to the longer datapath (Section IV-D). The clustering removes such overheads by localizing the communication within a train box.

VII. Related Work

Here, we explain the related works. Refer to Section II-B for the model computation and synchronization overheads.



(a) Image

(b) Audio

Fig. 22: Host-side resource utilization of various server architectures for neural network training. We normalized the resource consumption to the consumption of the baseline.

A. Data Preprocessing Accelerators

Our TrainBox can leverage existing data processing accelerators to provide data preparation functionalities for various input forms. For example, there exist decoding accelerators for JPEG [28], PNG [17], and JPEG 2000 [10]. Similarly, the graphics domain can take advantage of hardware-based 2D DCT operations [45], and video decoding [38]. With these implementations, TrainBox can handle different input forms by exploiting the partial reconfiguration at runtime [49].

B. Data Augmentation

Lastly, we introduce some data augmentation techniques from machine learning communities. Besides the basic data augmentation (e.g., crop, flip, noise addition) many works have proposed advanced data augmentation methods. For example, Perez et al. [37] explored various data augmentation methods and showed their high impacts on accuracy. Takahashi et al. [43] proposed an efficient cropping algorithm that randomly crops four images and merges them to create a new training image. For the audio domains, SpecAugment [35] discusses applying augmentation techniques on the Mel spectrogram. We expect that more data augmentation methodologies will emerge. In this case, TrainBox will play an important role by reducing the data augmentation overhead while preserving their benefits.

VIII. Conclusion

We proposed TrainBox, a neural network server architecture to enable scalable training. With the advent of highly efficient accelerators and synchronization methodologies, we observed that the performance bottleneck will shift from model computation and synchronization to data preparation. To maximize the hardware acceleration benefits, we focused on the data preparation bottleneck and proposed TrainBox with three novel optimizations. TrainBox successfully reduced the CPU, memory bandwidth, and PCIe bandwidth consumption by freeing the host from data preparation and achieves high scalability. We envision that TrainBox's importance will increase with better neural network accelerators and emerging data augmentation techniques.

Acknowledgment

This work was partly supported by Samsung Electronics, National Research Foundation of Korea (NRF) grant funded by the Korean Government (NRF-2015M3C4A7065647, NRF-2017R1A2B3011038, NRF-2019R1A5A1027055, NRF-2020M3H6A1084857), Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean Government (No. 1711080972), and Creative Pioneering Researchers Program through Seoul National University. We also appreciate the support from Automation and Systems Research Institute (ASRI), Inter-university Semiconductor Research Center (ISRC) and Neural Processing Research Center (NPRC) at Seoul National University.

References

[1] J. Ahn, D. Kwon, Y. Kim, M. Ajdari, J. Lee, and J. Kim, "Dcs: a fast and scalable device-centric server architecture," in *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*, 2015.

[2] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications," in *Proceedings of the 2018 International Solid-State Circuits Conference (ISSCC)*, 2018.

[3] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim *et al.*, "A cloud-scale acceleration architecture," in *Proceedings of the 49th International Symposium on Microarchitecture (MICRO)*, 2016.

[4] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

[5] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[6] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, 2016.

[7] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, 2016.

[8] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[10] A. Descampe, F.-O. Devaux, G. Rouvroy, J.-D. Legat, J.-J. Quisquater, and B. Macq, "A flexible hardware jpeg 2000 decoder for digital cinema," *IEEE Transactions on Circuits and Systems for Video Technology (CAS)*, vol. 16, no. 11, pp. 1397–1410, 2006.

[11] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," in *CVPR Workshops*, 2011.

[12] "Google cloud tpu," Google, Accessed Sep. 2, 2020. [Online]. Available: https://cloud.google.com/tpu/

[13] 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.

[14] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A gpu cluster manager for distributed deep learning," in *Proceeding of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2019.

[15] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *Proceeding of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[16] B. Hong, Y. Ro, and J. Kim, "Multi-dimensional parallel training of winograd layer on memory-centric architecture," in *Proceedings of the 51st International Symposium on Microarchitecture (MICRO)*, 2018.

[17] S. Huang and T. Zheng, "Hardware design for accelerating png decode," in *Proceedings of the International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, 2008.

[18] "Intel rack scale design architecture," Intel, Accessed Mar. 5, 2020. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/rack-scale-design-architecture-white-paper.pdf

[19] "Intel® xeon phi™ x200 product family," Intel, Accessed Mar. 5, 2020. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/series/92650/intel-xeon-phi-x200-product-family.html

[20] S. Jeaugey, "Massively scale your deep learning training with nccl 2.4," Accessed Sep. 02, 2020. [Online]. Available: https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/

[21] Y. Ji, Y. Zhang, X. Xie, S. Li, P. Wang, X. Hu, Y. Zhang, and Y. Xie, "Fpsa: A full system stack solution for reconfigurable reram-based nn accelerator architecture," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the International Conference on Multimedia (Multimedia)*, 2014.

[23] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th International Symposium on Computer Architecture (ISCA)*, 2017.

[24] D. Kwon, J. Ahn, D. Chae, M. Ajdari, J. Lee, S. Bae, Y. Kim, and J. Kim, "Dcs-ctrl: a fast and flexible device-control mechanism for device-centric server architecture," in *Proceedings of the 45th International Symposium on Computer Architecture (ISCA)*, 2018.

[25] Y. Kwon and M. Rhu, "Beyond the memory wall: A case for memory-centric hpc system for deep learning," in *Proceedings of the 51st International Symposium on Microarchitecture (MICRO)*, 2018.

[26] Y. Li, J. Park, M. Alian, Y. Yuan, Z. Qu, P. Pan, R. Wang, A. Schwing, H. Esmaeilzadeh, and N. S. Kim, "A network-centric hardware/algorithm co-design to accelerate distributed training of deep neural networks," in *Proceedings of the 51st International Symposium on Microarchitecture (MICRO)*, 2018.

[27] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[28] M. Mody, V. Paladiya, and K. Ahuja, "Efficient progressive jpeg decoder using jpeg baseline hardware," in *Proceedings of the International Conference on Image Information Processing (ICIIP)*, 2013.

[29] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *Proceedings of the 2017 International Solid-State Circuits Conference (ISSCC)*, 2017.

[30] "Nvidia collective communications library (nccl)," NVIDIA, 2017.

[31] "Dgx2 training performance," NVIDIA, Accessed Feb. 08, 2020. [Online]. Available: https://developer.nvidia.com/deep-learning-performance-training-inference

[32] "NVIDIA DALI," NVIDIA, Accessed Feb. 08, 2020. [Online]. Available: https://docs.nvidia.com/deeplearning/sdk/dali-developer-guide/docs/index.html

[33] "Nvidia dgx-2 the world's most powerful deep learning system for the most complex ai challenge," NVIDIA, Accessed Sep. 2, 2020. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-1/dgx-2-datasheet-us-nvidia-955420-r2-web-new.pdf

[34] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[35] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *Proceedings of the 20th Conference of the International Speech Communication Association (INTERSPEECH).*, 2019.

[36] "Pci express® base specification revision 3.0," PCI SIG, 2010.

[37] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[38] G. Plagne, "Efficient video decoding accelerator," 2008, uS Patent App. 11/912,007.

[39] A. Sanaullah and M. C. Herbordt, "Fpga hpc using opencl: Case study in 3d fft," in *Proceedings of the International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2018.

[40] F. Serzhenko and V. Podlozhnyuk, "Fast jpeg coding on the gpu," in *GPU Technology Conference*, 2011.

[41] Supermicro, "Superserver 4029gp-trt2," Accessed Sep. 02, 2020. [Online]. Available: https://www.supermicro.com/en/products/system/4U/4029/SYS-4029GP-TRT2.cfm

[42] O. S. Systems, "4u value 16-slot expansion system," Accessed Sep. 02, 2020. [Online]. Available: https://www.onestopsystems.com/product/4u-value-16-slot-expansion-system

[43] R. Takahashi, T. Matsubara, and K. Uehara, "Data augmentation using random image cropping and patching for deep cnns," *IEEE Transactions on Circuits and Systems for Video Technology (CAS)*, 2019.

[44] P. Technology, "Pex8796," Accessed Sep. 02, 2020. [Online]. Available: http://static6.arrow.com/aropdfconversion/b2e6d0c248399662feaaa658cc49bab2aa8e6930/372448.pdf

[45] A. Tumeo, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, "A pipelined fast 2d-dct accelerator for fpga-based socs," in *Proceedings of the Symposium on VLSI (ISVLSI)*, 2007.

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2017.

[47] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of the 46th International Symposium on Microarchitecture (MICRO)*, 2013.

[48] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: Lessons learned from the 2015 mscoco image captioning challenge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 652–663, 2016.

[49] "Vivado design suite user guide: Partial reconfiguration," Xilinx, Accessed Feb. 08, 2020. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug909-vivado-partial-reconfiguration.pdf

[50] "Vivado high-level synthesis: Accelerates ip creation by enabling c, c++ and system c specifications," Xilinx, Accessed Feb. 08, 2020. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[51] "Xilinx virtex ultrascale+ fpga vcu1525 acceleration development kit," Xilinx, Accessed Feb. 08, 2020. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/vcu1525-a.html