CANNON: Communication-Aware Sparse Neural Network Optimization

A. Alper Goksoy[1], Guihong Li[2], Sumit K. Mandal[3], Umit Y. Ogras[1], Radu Marculescu[2]

Abstract—Sparse deep neural networks (DNNs) have the potential to deliver compelling performance and energy efficiency without significant accuracy loss. However, their benefits can quickly diminish if their training is oblivious to the target hardware. For example, fewer critical connections can have a significant overhead if they translate into long-distance communication on the target hardware. Therefore, hardware-aware sparse training is needed to leverage the full potential of sparse DNNs. To this end, we propose a novel and comprehensive communication-aware sparse DNN optimization framework for tile-based in-memory computing (IMC) architectures. The proposed technique, CANNON first maps the DNN layers onto the tiles of the target architecture. Then, it replaces the fully connected and convolutional layers with communication-aware sparse connections. After that, CANNON optimizes the communication cost with minimal impact on the DNN accuracy. Extensive experimental evaluations with a wide range of DNNs and datasets show up to 3.0× lower communication energy, 3.1× lower communication latency, and 6.8× lower energy-delay product compared to state-of-the-art pruning approaches with a negligible impact on the classification accuracy on IMC-based machine learning accelerators.

Index Terms—Hardware-aware pruning, communication-aware pruning, mapping, sparse neural networks.

1 Introduction

Deep neural networks (DNNs) exhibit a high degree of redundancy due to dense interconnections between successive layers. Besides posing overfitting risks, redundant connections increase the communication cost and implementation overhead, thus leading to lower performance and energy efficiency when implemented in hardware. Indeed, many pruning techniques aim at removing DNN connections with minimal impact on their accuracy [1], [2], [3]. Sparse neural networks are preferred since they can enable minimal communication and implementation overhead, thus significantly reducing the computation and memory requirements.

Sparse inter-layer connections enable significantly faster and more energy-efficient DNNs. However, sparsity alone is *not* sufficient since good algorithmic performance *does not* necessarily translate into real performance on hardware. For instance, some inter-layer connections can lead to long paths when mapped on hardware. Consequently, they can undermine the overall hardware performance due to high communication latency and energy costs. For example, the sparse evolutionary training (SET) approach [4] drastically decreases the training time using sparse graphs instead of pruning a trained network; this makes the training scalable, while improving the test accuracy on a wide range of datasets, including multi-layer perceptron (MLP) and convolutional neural networks (CNNs) for unsupervised and supervised learning. Although it can achieve higher accuracies, the networks remain oblivious to the real hardware. The performance of DNNs on real hardware is critical since it determines the inference latency and power consumption. For example, a DNN targeting real-time applications, such as autonomous driving, may become impractical if the inference latency violates the timing constraints. To analyze the inference latency, we perform experiments using an in-memory computing (IMC)-based DNN accelerator where the inter-layer communication for activation data movement is implemented via a network-on-chip (NoC). We use a state-of-the-art reinforcement learning based mapping algorithm [5] with unpruned networks using two different datasets. Our evaluations show that the communication between the DNN layers alone can take up to 43% of the total inference latency for a wide range of DNNs, as depicted in

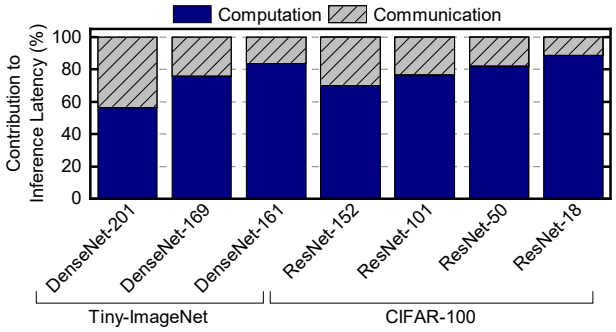


Fig. 1. Percentage contribution to inference latency for various networks on two datasets. The communication latency can take up to 43% of the total inference latency.

[1]*A. Alper Goksoy, and Umit Y. Ogras are with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison 53706, WI;*

[2]*Guihong Li and Radu Marculescu are with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin 78712, TX.*

[3]*Sumit K. Mandal is with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, Karnataka.*

This work was supported in part by the US National Science Foundation (NSF) grant CNS-2007284, and in part by Semiconductor Research Corporation (SRC) grants GRC 2939.001 and 3012.001. *E-mail: {agoksoy, uogras}@wisc.edu, {lgh, radum}@utexas.edu, skmandal@iisc.ac.in*

Figure 1.

Although sparse training can achieve a higher accuracy than a network with no pruned links [6], if the network remains oblivious to the target hardware while pruning and adding links, then, this can lead to unacceptable latency and power overheads. Therefore, there is a strong need for *hardware and communication-aware sparse training* methodologies that can lead to shorter communication distances when the DNN layers are mapped onto hardware resources.

Starting from these observations, this paper presents CANNON, a novel **c**ommunication-**a**ware sparse **n**eural **n**etwork **o**ptimizatio**n** technique applicable to both fully connected and convolutional layers. The first step of the proposed technique maps the DNN layers on the target hardware resources, e.g., to the processing tiles of an NoC. Our proposed mapping technique minimizes the distance the packets between two consecutive DNN layers need to travel in the NoC which helps reducing the overall communication latency. The second step performs hardware-aware dynamic sparse training. Suppose two nodes in the DNN are connected by links with non-zero weights. If these DNN nodes are mapped onto different tiles on the NoC, the activations generated between these nodes will incur communication costs during inference. The proposed technique prunes the $p-$percent of the weights based on the significance of weight columns (called the *z-index*) towards any inference decision per unit communication cost. Finally, we maintain the target sparsity throughout the training process by choosing an equal number of weight columns ($p-$percent) with the smallest communication cost and adding them back to the network at the end of each epoch.

We evaluate CANNON exhaustively using well-established simulators (NeuroSim [7], BookSim [8]), popular DNN structures (ResNet [9], DenseNet [10], VGG-16 [11], MLP), and datasets (Tiny-ImageNet [12], CIFAR-100, CIFAR-10 [13], MNIST [14]). The hardware performance of the sparse neural networks with our proposed technique is also compared against state-of-the-art pruning techniques [4], [6]. Our hardware-optimized sparse neural networks result in up to 6.8× improvement in the energy-delay product (with respect to the neural networks with pruning and state-of-the-art mapping techniques), without any significant accuracy degradation.

The major contributions of this work are as follows:

- A latency-aware mapping technique which minimizes the distance packets between DNN layers travel between processing elements using an NoC;
- A hardware-aware pruning technique using a newly proposed *z-index* that guarantees sparsity while further reducing the communication latency;
- Extensive experimental evaluations showing 1.6×–3.1× latency and 2.7×–6.8× energy-delay product improvements compared to state-of-the-art pruning techniques without a significant accuracy loss.

The rest of the paper is organized as follows. Prior work related to DNN pruning and hardware mapping are discussed in Section 2. The proposed technique is described in detail in Section 3. The experimental results are presented in Section 4. Finally, Section 5 concludes the paper.

TABLE 1
Comparison of various mapping and pruning approaches

Method	FC Pruning	Conv Pruning	HW-Aware Training	Mapping
Wu et al. [5]	no	no	no	yes
Mocanu et al. [4]	yes	no	no	no
Frankle & Carbin [6]	yes	yes	no	no
Karimzadeh et al. [19]	yes	yes	no	yes
Chu et al. [20]	yes	yes	no	yes
Meng et al. [21]	yes	yes	no	yes
CANNON	**yes**	**yes**	**yes**	**yes**

2 Related Work and Novel Contributions

Network pruning is a widely studied approach to reducing network sizes by eliminating redundant parameters. Optimal Brain Damage [15] and Optimal Brain Surgeon [16] are early works in this domain. Authors in [17] prune the network weights with an absolute value closest to zero and a little to no contribution to the output. More recent pruning methods incorporate training to retain a similar accuracy as the original network [18]. For example, the Lottery Ticket Hypothesis (LTH) finds a subnetwork, referred to as the winning ticket, that can achieve the same test accuracy as the unpruned network if trained in isolation [6]. The authors train a network with randomly initialized weights for a predefined number of epochs. Then, p-percentage of the weights with the smallest absolute value is pruned to obtain the winning ticket. After that, the unpruned weights are reset to their initial values before repeating the training. Then, the pruned network, referred to as the *winning ticket* subnetwork, is trained for the same number of epochs. By iteratively repeating this process, the authors generate winning tickets that have 80%–90% fewer weights while keeping a similar accuracy. Table 1 shows the comparison of CANNON against LTH and other the state-of-the-art pruning and mapping methods.

Similar to network pruning, a sparse evolutionary training approach, called SET, can guarantee built-in sparse structures during training [4]. SET achieves higher test accuracies compared to unpruned networks. However, it uses sparse connections only for the fully connected layers, leaving the convolutional layers intact.

None of the mentioned approaches considers the hardware resources and accounts for the communication cost during pruning. Hence, these approaches *cannot* exploit the full potential of the target hardware. Recent studies started considering hardware-aware techniques due to the importance of the target hardware. Wu et al. [5] use Reinforcement Learning (RL) for mapping DNNs to hardware, but they do not perform pruning. Wang et al. [22] quantize DNNs to reduce the model size using a hardware-aware quantization method. A Linear Feedback Shift Register is used to decide which weights to prune (or not) in [19]. Lately, structural pruning gained popularity due to its pruning in regular shapes. Structural pruning prunes the network such that a significant chunk of memory elements can be removed. Structural pruning techniques for CPUs are proposed in [23], [24]. Similarly, the Scalpel technique [25] prunes networks to match the network size to SIMD units. 3PXNet [26] combines binarization and pruning for edge

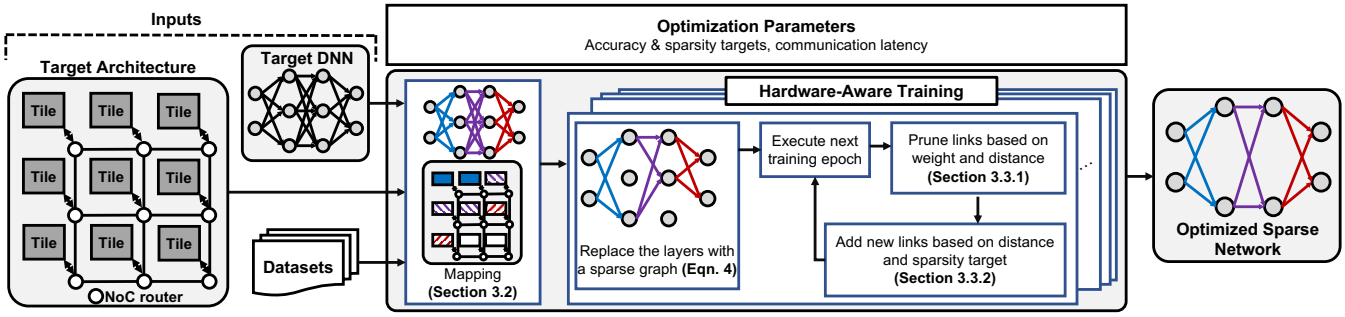


Fig. 2. Overview of the proposed approach, CANNON. It consists of mapping the target DNN onto the target architecture using the latency-aware mapping and hardware-aware dynamic sparse training. The training process first replaces the DNN layers with sparse graphs; then, at the end of each epoch, employs hardware-aware pruning and link addition. Each circle in the target DNN represents the feature map of DNNs; each link in the target DNN represents the weights of DNNs. The weights are mapped onto the in-memory computing (IMC) tiles with the same color as the corresponding links. The circles and the rectangles in the target architecture denote the NoC routers and IMC tiles, respectively.

machine learning. Technique in [2] proposes to remove a complete channel filter of a convolutional network instead of randomly selecting the weights in a channel to prune on embedded GPUs. However, the authors show that existing GPU libraries do not perform channel pruning efficiently. The methods proposed in [3], [20], [21], [27], [28], [29], [30], [31] use resistive-RAM (RRAM) based hardware accelerators for structured pruning and propose crossbar-aware structural pruning to prune the network. Authors in [20], [21], [27], [29], [30] employ crossbar-aware column and row pruning, which necessitates an indexing unit to handle migrating weight columns. This hardware unit results in extra overhead. Authors in [28] do not use indexing units but reduce the number of bits required for the ADC. Authors in [3], [31] use operation units that divide the crossbar arrays into small matrices to utilize each crossbar column with multiple weight columns. This approach utilizes the time division principle (i.e., activates some part of a crossbar column at a particular time instance) for the weight columns, resulting in an extra time overhead. However, any of the mentioned approaches do not consider the overhead of the communication of activations on the system while utilizing structured pruning.

The unique aspects of our work are highlighted in Table 1 where we compare CANNON against the state-of-the-art network pruning and mapping methods. In contrast to prior techniques, we account for the communication cost to enable hardware-aware training. We first map the target network onto the target hardware to minimize the communication latency and then perform hardware-aware dynamic sparse training. Our proposed methodology can work with any pruning technique. We chose SET [4] as the baseline since it guarantees sparse connections between each DNN layers by construction. However, different than SET, our approach also prunes convolutional (in addition to the fully connected) layers. Our systematic approach introduces sparse graphs that can enable structured sparsity where blocks of zeros can be obtained. Hence, our proposed technique complements efficient sparse representations like Compressed Sparse Block (CSB) [32], Compressed Sparse Row (CSR) [33], Compressed Sparse Fiber (CSF) [34] to minimize data communication overheads. To the best of our knowledge, CANNON is the first hardware-aware training technique that combines all the features listed in Table 1.

3 Methodology

In this section, we first overview the proposed approach. Then, we present our latency-aware mapping and hardware-aware dynamic sparse training in detail. Finally, we illustrate the evolution of the z-index.

3.1 Overview of CANNON

The inputs to our framework are the target DNN, the datasets, and the hardware architecture, as shown in Figure 2. Our goal is to prune the DNN to meet a given sparsity target which maintains the accuracy of the DNN, while minimizing communication latency on the target hardware. To this end, we first map the DNN nodes to the target hardware as described in Section 3.2. Then, we perform the newly proposed hardware-aware training. At the beginning of training, we replace the layers of the DNN with a sparse graph. Then, we perform hardware-aware pruning, as well as hardware-aware link addition during each epoch as described in sections 3.3.1 and 3.3.2, respectively. At the end of the training process, we obtain a hardware-aware sparse network and its mapping onto a mesh NoC.

3.2 Latency-Aware Mapping

The first step of CANNON is to map the nodes of the target DNN onto the target hardware, which we assume is an in-memory computing (IMC)-based DNN accelerator. We consider IMC-based DNN accelerators since they integrate computation with memory, and decrease the latency and energy cost of memory accesses. Several recent research have proposed energy-efficient DNN accelerators with IMC technology [35], [36], [37], [38]. IMC accelerators integrate multiple processing elements (known as tiles) into the system, as shown in Figure 3. The number of tiles in the system is a function of neural network parameters, as well as the hardware parameters. In this work, the workload is divided into tiles following the technique described in [7]. Each tile, placed on a grid, consists of memory elements that store the DNN weights and perform computations. Finally, the tiles are connected to NoC routers which facilitate the data exchanges between different neural network layers. The CANNON framework maps the DNN into the IMC accelerator layer-by-layer. To this end, we analyze the traffic

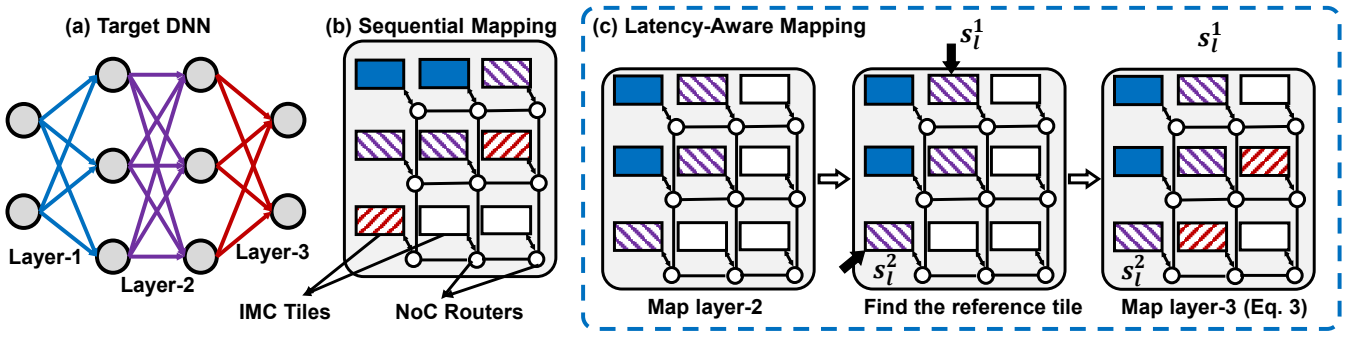


Fig. 3. Illustration of latency-aware mapping algorithm. (a) the target DNN to be mapped, (b) sequential mapping [37] maps the layers of the target DNN to the tiles of NoC sequentially using the number of tiles required for each layer, (c) latency-aware mapping algorithm in CANNON first finds the reference tiles (s_l^1, s_l^2) of previously mapped layers and then maps the next layer by minimizing the total distance to the reference tiles. **All rectangular boxes in (b) and (c) represent IMC tiles that contain processing elements while the small circles represent NoC routers.**

between any two consecutive layers of the neural network to map their nodes. We note that the communication latency between two consecutive network layers is mainly determined by the position of tiles the nodes in these layers are mapped to. Figure 3 shows an illustrative example of mapping a target DNN (Figure 3(a)) onto tiles. The sequential mapping places the DNN nodes onto the IMC tiles in order, from left to right and from top to bottom [37]. With this mapping style, some of the packets may need to travel long distances. For example, in Figure 3(b), there is communication between the tile in the top right corner to the tile in the bottom left corner.

In contrast to prior work [37], we compute the distance between any s (source) and d (destination) tiles before mapping as $M(s,d)$:

$$M(s,d) = |x_s - x_d| + |y_s - y_d| \tag{1}$$

where x_s (or x_d) denotes the physical x-coordinate of the tile-s (or tile-d) and y_s (or y_d) denotes the physical y-coordinate of the tile-s (or tile-d). Then, to ensure that the DNN nodes in two consecutive layers are not physically far apart from each other, we minimize the maximum distance between the tiles. Hence, Equation 2 represents the objective of our mapping technique:

$$\text{minimize} \quad \max_{i,j} M(s_i, d_j), \quad 1 \le i \le T_l, 1 \le j \le T_{l+1}, \tag{2}$$

where, T_l denotes the number of tiles in the l^{th} layer.

For a system with K tiles, the complexity to solve Equation 2 is $O(T_{l+1}(K - T_{l+1}))$. To reduce this complexity, we consider a pair of source tiles (s_l^1, s_l^2) that are physically farthest from each other, instead of considering all the source tiles (tiles corresponding to the l^{th} layer in Equation 2).

Figure 3(c) illustrates the proposed latency-aware mapping used in CANNON. Assume that there are K IMC tiles, and each tile is connected to an NoC router. At the beginning of the mapping process, we map the first layer of the neural network to the IMC tiles closest to the input of the accelerator (the blue tiles in Figure 3(c)). Since T_l denotes the number of tiles required for l^{th} layer, the remaining layers that are mapped after l^{th} layer are $R_l = K - \sum_{i=l+1}^{L} T_i$, where L is the total number of DNN layers. To map the $(l+1)^{\text{th}}$ layer, we calculate the total distance from each of the remaining tiles (R_l) to s_l^1 and s_l^2 (highlighted with solid arrows in Figure 3(c)) following Equation 1. Then, the tile that minimizes the sum of the distance to the reference tiles is selected. Specifically, the position of this tile is:

$$\underset{j}{argmin}(M(s_l^1, d_j) + M(s_l^2, d_j)), 1 \le j \le R_l \tag{3}$$

The above process is repeated for *each* layer until all the layers are mapped. We note that our mapping algorithm always finds a solution if $\sum_{l=1}^{L} T_l \le K$.

We note that our proposed mapping technique is independent of the traffic volume exchanged between different DNN layers. Specifically, we consider only two consecutive layers at a time. We also note that the proposed mapping technique takes care of all existing types of connections in a neural network, namely linear, skip, and dense connections. Therefore, our proposed mapping technique is applicable to *any DNN*, irrespective of traffic volume and connection pattern. The detailed discussion on how the proposed mapping technique is applied to a neural network with skip and dense connections is presented in Appendix A.

3.3 Hardware-Aware Dynamic Sparse Training

3.3.1 *Hardware-Aware Pruning*

Our hardware-aware pruning approach uses the Sparse Evolutionary Training, i.e., SET, technique as a baseline to guarantee sparse connections between each layer in the training process [4]. It follows a phenomenon observed in complex real-world networks such as protein interaction networks. This phenomenon shows that starting with an Erdös–Rényi random graph [39] and following its natural evolution, the network reaches a point that has a more structured connectivity resembling scale-free [40] or small-world [41] networks. Inspired by this phenomenon, at the beginning of the training process, the fully connected layers in the DNN are replaced by sparsely connected layers following Erdös–Rényi random sparse graphs as in Equation 4:

$$r(\mathcal{W}_i) = \frac{\epsilon(n_i + n_{i-1})}{n_i n_{i-1}} \tag{4}$$

where $\mathcal{W}_i$ represents the set of weights in layer-i, where $1 \le i \le L$ and L represents the number of layers in the target DNN. n_i and n_{i-1} represent the number of weights in layer-i and layer-$(i-1)$, respectively. $\epsilon \in \mathbb{R}^+$ is a tunable parameter used to adjust for the target sparsity level [4],

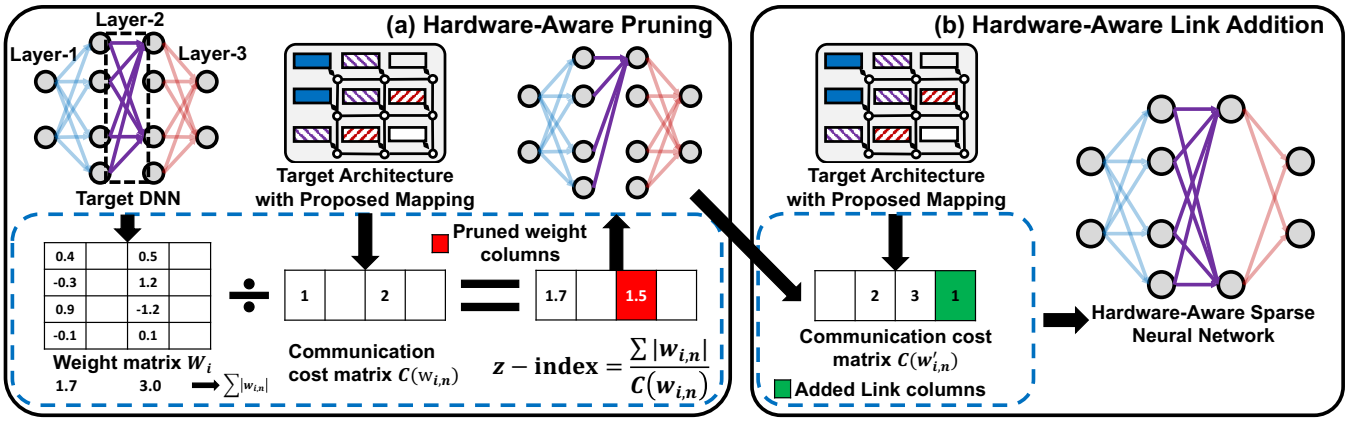


Fig. 4. Proposed hardware-aware pruning and link-addition. (a) In hardware-aware pruning, z-index of each weight column is computed based on the absolute value and the communication cost. Then, the p-percentage of weight columns with the lowest z-index is removed. (b) The communication cost of already pruned weight columns is calculated in the hardware-aware link addition step. Then, the p-percentage of weight columns with the lowest communication cost is added back. This procedure is repeated every epoch for each layer. The numbers in weight and communication cost matrices are shown for illustration purposes.

while $r(\mathcal{W}_i)$ represents the probability that each weight in the set is retained. Our hardware-aware pruning approach is very general, as it works for both fully connected (FC) and convolutional (Conv) layers. For layer-i, we define the depth of the input feature channels as D_i, the kernel depth as N_i, and the kernel size as K_i. We can organize the weights of the FC and Conv layers as matrices of sizes $D_i \times N_i$ and $K_i \times K_i \times D_i \times N_i$, respectively. Therefore, we can represent the n^{th} weight column of layer-i as $w_{i,n} \in W_i$ where $1 \leq n \leq N_i$. Each column of the weight matrix ($D_i \times 1$ for FC, $K_i \times K_i \times D_i \times 1$ for Conv layer) is mapped to a corresponding column of the crossbar array. During execution, each input feature map block (the same size as the weight matrix column) is multiplied with each crossbar column, generating output feature map blocks ($1 \times N_i$). By working at the level of individual columns in the weight matrix, our approach results in pruned columns rather than producing an unstructured sparse weight matrix.

Our goal at the end of each training epoch is to prune the DNN weights that incur a significant communication latency without losing significant accuracy, as illustrated in Figure 4. Each weight column of the DNN generates activation(s) communicated between any two DNN layers. Therefore, pruning weights implicitly leads to removing some communication between any two DNN layers.

Definition 3.1 (Communication Cost). The contribution of a weight column n for layer-i ($w_{i,n} \in \mathcal{W}_i$) to the communication can be quantified by the *average Manhattan distance* ($M(s,d)$) between the source tile (s) and the destination tile (d) of the corresponding activations:

$$C(w_{i,n}) = \frac{\sum_{j=1}^{A_{i,n}} M(s_i^j, d_i^j)}{A_{i,n}} \tag{5}$$

where $A_{i,n}$ is the number of activations generated by the corresponding weight column $w_{i,n}$ and $M(s,d)$ is given in Equation 1. However, the significance of weights also depends on their total absolute value. Thus, pruning weights should be based on both weights' absolute value and corresponding communication cost due to their activation.

Definition 3.2 (z-index). z-index is the ratio between the total absolute value of a weight column and its communication cost:

$$z(w_{i,n}) = \frac{\sum(|w_{i,n}|)}{C(w_{i,n})} \tag{6}$$

The z-index reflects the *importance* of a weight column in terms of both absolute value and the communication cost, as defined in Equation 6. Note that, a weight column with a higher total absolute value will have a higher z-index, while a weight column with higher communication cost will have a lower z-index. Hence, the z-index can be interpreted as the *weight's significance* per unit of communication cost. Therefore, we use the z-index of each weight column to determine whether to prune it. Specifically, we sort all weight columns in layer-i based on their z-index values and prune the smallest p-percentage of weight columns, where p represents the target pruning ratio.

3.3.2 Hardware-Aware Link Addition

After pruning weights during each epoch, an equal number of weights are added to the DNN to maintain the initial pruning ratio. In SET [4], these weights are added randomly. In contrast, *we add new weights considering the communication cost* ($C(w'_{i,n})$), where $w'_{i,n} \in \mathcal{W}'_{i,n}$ and $\mathcal{W}'_{i,n}$ denotes the set of already pruned weight columns from layer-i. Then, we sort $C(w'_{i,n})$ and add p-percentage of weights in $\mathcal{W}'_{i,n}$ with the lowest communication cost as shown in Figure 4(b). This way, the performance gains from latency-aware mapping and hardware-aware pruning are preserved by considering the communication cost in the link addition process. As a result, we retain the same number of weights throughout the training process.

3.4 Evolution of the Average z-Index

In this section, we analyze the evolution of the average z-index of all weights in the DNN during training. This analysis provides invaluable insights into how the proposed hardware-aware pruning and hardware-aware link addition techniques work. We use the ResNet-50 network on the

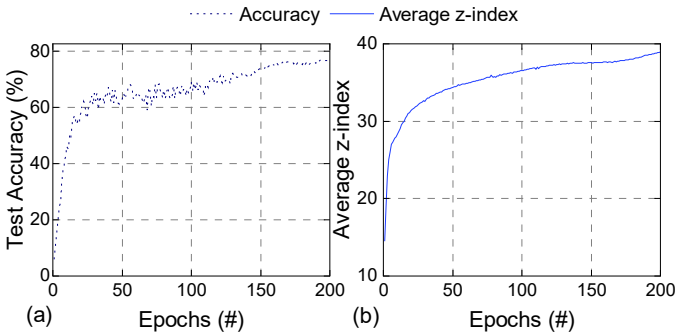


Fig. 5. Illustration of the evolution of the (a) test accuracy and (b) the average z-index of all weight columns during training for ResNet-50 on the CIFAR-100 dataset. The accuracy, as well as the average z-index, increases throughout the training. Increasing z-index denotes a DNN with larger weights and lower communication latency due to our hardware-aware dynamic sparse training approach.

CIFAR-100 dataset for illustration, but we note that all models and datasets considered in our experimental evaluations show similar trends.

The accuracy and z-index variations for each epoch in the training are shown in Figure 5(a) and Figure 5(b), respectively. The average z-index grows similar to the accuracy for two reasons. First, the weights (the numerator of the z-index) that are preserved are larger than those pruned in each epoch. Second, hardware-aware pruning and weight addition favor smaller communication costs (the denominator of the z-index). Indeed, Figure 5(b) confirms that the average z-index grows rapidly during training and follows a similar trend to the classification accuracy. Specifically, the average z-index is small at the beginning of the training. Our proposed hardware-aware pruning process removes the weights with lower absolute value and that causes higher communication costs during training. Therefore, the average z-index increases over time. At the end of Epoch-200 (when the accuracy stabilizes), the average z-index is 3.2$\times$ larger than the average z-index in the beginning. The growth seen in the average z-index indicates a DNN with lower communication latency. Hence, it improves the hardware performance, as discussed in the following section.

4 Experimental Evaluation

This section first introduces our experimental setup to enable reproducible results. Next, we demonstrate the effectiveness of our proposed latency-aware mapping and hardware-aware dynamic sparse training approaches in terms of the number of hops distribution in NoC. Then, we compare the hardware performance of CANNON in terms of latency, energy, energy-delay product (EDP), and accuracy against state-of-the-art techniques. After that, we perform an ablation study of CANNON. Finally, we compare our results against Lottery Ticket Hypothesis using networks from ResNet family on the CIFAR-10 dataset.

4.1 Experimental Setup

Network Models and Datasets: We evaluate CANNON with 14 well-known network models and four datasets. Specifically, we use DenseNet-201, DenseNet-169, and DenseNet-161 from the DenseNet family with the Tiny-ImageNet dataset. Similarly, we employ ResNet-152, ResNet-101, ResNet-50, ResNet-18 and VGG-16 with CIFAR-100 and CIFAR-10 datasets. DenseNet, ResNet, and VGG-16 networks consist of convolutional and fully connected layers. We also discuss performance results using an MLP-based network on the MNIST dataset. We use the same MLP-based network structure used in *SET* [4] which consists of three hidden layers with 1000 neurons.

TABLE 2
Summary of circuit level and NoC parameters

Circuit		NoC	
PE array size	256 × 256	Bus width	32
Cell levels	2 bit/cell	Routing algorithm	X–Y
Flash ADC resolution	8 bits	Number of router ports	5
Technology used	RRAM	Topology	Mesh

Simulation Setup: The experiments run on a machine that uses 6 Intel Xeon Gold 6242R cores and 1 Nvidia 3090 GPU on Python using the PyTorch library. All CNNs are trained for 200 epochs with the SGD optimizer and a momentum of 0.9. We set the initial learning rate as 0.1 and use the Cosine Annealing scheduling as the learning rate scheduler. The communication performance is evaluated using a cycle-accurate NoC simulator, BookSim [8]. To this end, we use a customized version of BookSim that supports simulation with workload traces. As different networks show different structures, we generate traces for each network and dataset pairs. Each trace consists of three entries: source router, destination router, and timestamp for the generated packet. We generate a trace file for each layer and feed this to BookSim to measure the communication performance. The number of IMC tiles required for each layer of the neural network is evaluated through NeuroSim [7]. In the IMC tile structure, adopted from [37], there are 16 PEs; each PE consists of a 256×256 IMC crossbar. We design separate accelerators to show the effectiveness of our hardware-aware mapping and training approaches on each dataset and network model and assume that the accelerator is big enough to accommodate the network as in [28]. In this work, we do not consider a pipelined architecture since such architectures may result in pipeline stalls [42]. Moreover, pipelining incurs an extra area overhead due to the extra control logic required. Therefore, our experimental evaluations report the overall end-to-end communication latency and energy when there is no layer-to-layer pipelining. Table 2 shows different hardware parameters incorporated in our evaluations.

Baseline Approaches: We compare CANNON against multiple baseline approaches summarized in Table 3. The first baseline technique is a RL-based mapping algorithm [5] working on an unpruned network (*Mapping Only* in Table 3). This baseline is added to the comparison set to assess the impact of the mapping algorithm alone. SET [4] is utilized in two baselines. The first one uses SET with a widely-used mapping algorithm [37] (*SET* in Table 3). In contrast, the second one uses SET with our proposed latency-aware mapping to show the performance of our proposed mapping

TABLE 3
Properties of different approaches with respect to the mapping method, Fully Connected (FC) layer pruning, and Convolutional (CONV) layer pruning.

Approach	Mapping	FC Pruning	CONV Pruning
Mapping Only [5]	RL	X	X
SET [4]	sequential [37]	weight-based	X
SET [4] + Mapping	latency-aware	weight-based	X
CANNON (FC Layers)	latency-aware	hardware-aware	X
CANNON (FC+CONV Layers)	latency-aware	hardware-aware	hardware-aware

method (*SET + Mapping* in Table 3). Two variants of the CANNON framework are evaluated in these experiments. We selected SET technique as the baseline pruning approach in the training. However, we emphasize that CANNON can be used in conjunction with any other pruning technique including state-of-the-art structured pruning techniques [3], [20], [21], [30]. We first show the results of our proposed dynamic sparse training when applied to the fully connected layers of neural networks (*CANNON (FC Layers)* in Table 3) since the approach proposed in SET prunes FC layers only.

Finally, we compare all the baselines against CANNON applied to both fully connected and convolutional layers (last row in Table 3). The pruning ratios in these experiments are 50% for the convolutional layers and 80% for the fully connected layers for ResNet, DenseNet, and VGG-16 networks selected based on the highest pruning value without seeing an important accuracy drop. For MLP-based networks, we use a pruning ratio of 95% to have a fair comparison against *SET*. The pruning and addition ratios in the hardware-aware dynamic sparse training are selected as 30% of the remaining weights. To quantify the overhead of the z-index calculations during training, we also compare CANNON against a random pruning scheme using identical pruning ratios. Our measurements indicate that CANNON has only 7% higher training time per epoch for the ResNet-18 model on the CIFAR-100 dataset. Once the training completes, CANNON does not introduce any additional inference overhead.

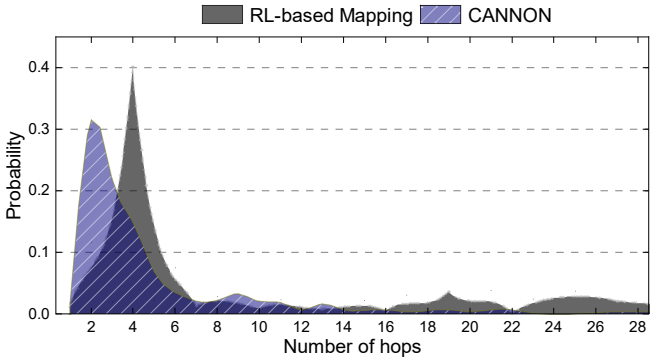


Fig. 6. Comparison of **hop distribution** between RL-based mapping [5] and CANNON. Latency-aware mapping minimizes number of hops probability toward smaller values.

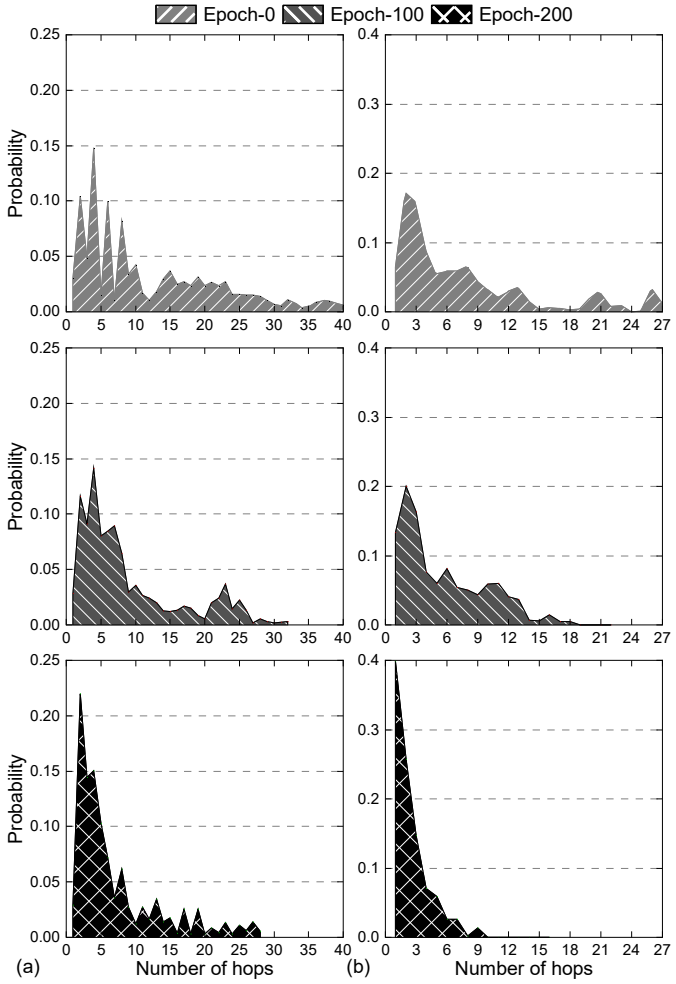


Fig. 7. Comparison of **hop distribution** for hardware-aware training before Epoch-0, at Epoch-100, and at the end of Epoch-200 for (a) ResNet-152 on the CIFAR-100 and (b) for DenseNet-201 on the Tiny-ImageNet dataset. The distribution of the number of hops shifts toward smaller numbers as we move forward during the training. There are still weights with higher hops at the end of Epoch-200 because the hardware-aware pruning considers communication cost and whether or not the weight is significant.

4.2 Number of Hops Distribution

CANNON aims at minimizing the communication latency by minimizing the maximum Manhattan distance the packets travel between two consecutive DNN layers (see Eqn. 2). Therefore, CANNON decreases the number of hops the packets travel compared to the state-of-the-art mapping technique [5]. Figure 6 compares the probability distribution of the number of hops the packets travel in the NoC for the DenseNet-201 model on Tiny-ImageNet in our proposed mapping technique and the technique described in [5]. Using the probability distributions, we observe that 86% of the packets need to traverse more than 3 hops in the NoC with the RL-based mapping approach. In contrast, the portion of the packets with more than 3 hops reduces to 45% with our proposed mapping technique.

The probability distribution for the number of hops for ResNet-152 model on CIFAR-100 and DenseNet-201 model on Tiny-ImageNet are shown in Figure 7(a) and Figure 7(b), respectively. These figures compare the distributions at the

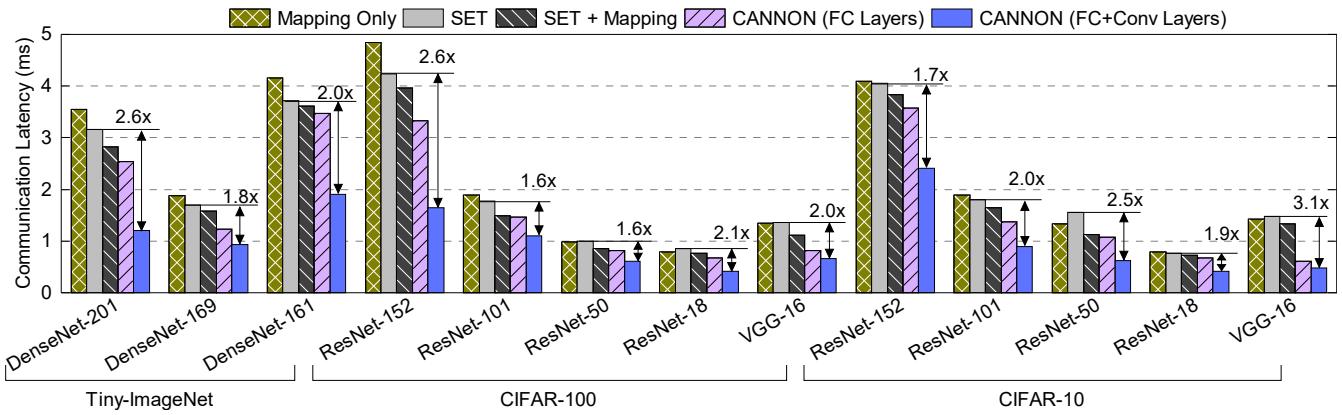


Fig. 8. Comparison of communication **latency** across different DNNs and datasets. CANNON consistently improves the latency for all network models and datasets.

beginning of the training (Epoch-0), at the end of Epoch-100, and at the end of training (Epoch-200). We observe that the probability of achieving a smaller number of hops increases with the number of epochs for the ResNet-152 model on CIFAR-100. Many long-range communications disappear as the training progresses due to our proposed hardware-aware pruning technique. We also analyze the percentage of weights distributed among different numbers of hops using probability distributions. Specifically, 44% of the packets need to traverse less than six hops at the beginning of Epoch-0; with our proposed hardware-aware dynamic sparse training, at the end of Epoch-200, 71% of the packets need to traverse less than six hops, as shown in Figure 7(a). However, we note that some long-range connections still remain because our hardware-aware pruning considers not only the communication cost but also the weight value. If there are weights with large values, we tend not to prune them even if they produce higher communication cost. Overall, the average number of hops decreases by 28% by the end of Epoch-100 and 42% by the end of Epoch-200 compared to Epoch-0. We observe a similar trend for the DenseNet-201 model on Tiny-ImageNet, where the probability of lower number of hops increases as the training progresses, as shown in Figure 7(b). The average number of hops decreases by 25% at the end of Epoch-100 and 70% at the end of Epoch-200. This is important because it translates into improved latency and energy efficiency.

4.3 Latency Analysis

This section compares the communication latency of CANNON against the four baseline techniques introduced earlier. The properties of each baseline are presented in Table 3. We use 14 widely-known network models listed in Section 4.1.

Experiments for convolutional neural networks are performed using DenseNet models on the Tiny-ImageNet dataset, ResNet and VGG-16 models using the CIFAR-100 and CIFAR-10 datasets, and an MLP using the MNIST dataset (Figure 8). The plot highlights the improvements of CANNON over the *SET* approach since a comparison against a pruned network is more appropriate.

DenseNet Network Results: Reinforcement Learning (RL)-based mapping approach using an unpruned DenseNet-201 network model on the Tiny-ImageNet dataset shows 3.55ms communication latency. Using *SET* [4] with the mapping in [37], the communication latency is improved to 3.16ms. The decrease is mainly due to the lack of some weight columns because of high pruning ratio. Our proposed latency-aware mapping employed with *SET* improves the communication latency by 20% compared to the RL mapping approach on an unpruned network for the DenseNet-201 model. The only difference between *SET* and *SET+Mapping* approaches is due to the mapping algorithm. Thus, the direct comparison between them demonstrates the effectiveness of our proposed latency-aware mapping. The improvement of *SET+Mapping* compared to *SET* ranges between 3% to 12% for the DenseNet models. *CANNON (FC Layers)* shows a speedup between $1.1\times$–$1.4\times$ compared to *SET*. Finally, *CANNON (FC+Conv Layers)* shows a higher speedup varying between $1.8\times$ to $2.6\times$ and $2.0\times$ to $3.0\times$ compared to *SET* and *Mapping Only* approaches for all networks in the DenseNet family, respectively. Considering the savings of computation, the improvements for total inference latency are up to 57% for DenseNet.

ResNet Network Results: The number of output channels of the convolutional layers of ResNet DNNs is higher than that of DenseNet DNNs. Therefore, the communication latency of ResNet DNNs is higher than DenseNet DNNs. The *Mapping Only* approach, which uses RL-based mapping on an unpruned network, shows the highest communication latency in most ResNet networks using the CIFAR-100 and CIFAR-10 datasets. The RL-based mapping performs better than the *SET* approach with ResNet-50 and ResNet-18 networks on CIFAR-100 and ResNet-50 on CIFAR-10 datasets. The *SET + Mapping* approach, which uses our proposed latency-aware mapping but pruning as proposed in SET, consistently outperforms *SET* and *Mapping Only* techniques. *CANNON (FC Layers)* further improves the latency between $1.1\times$ to $1.2\times$ compared to the *SET + Mapping* approach on the CIFAR-100 dataset. Finally, our proposed *CANNON (FC+Conv Layers)* shows a speedup between $1.6\times$ to $2.6\times$ compared to *SET* on CIFAR-100, as illustrated in Figure 8. On the CIFAR-10 dataset, the improvements are in the range of $1.7\times$ to $2.2\times$ compared against the *Mapping Only* approach. Communication latency improvements result in a 23%–51% drop in total inference latency for ResNet DNNs.

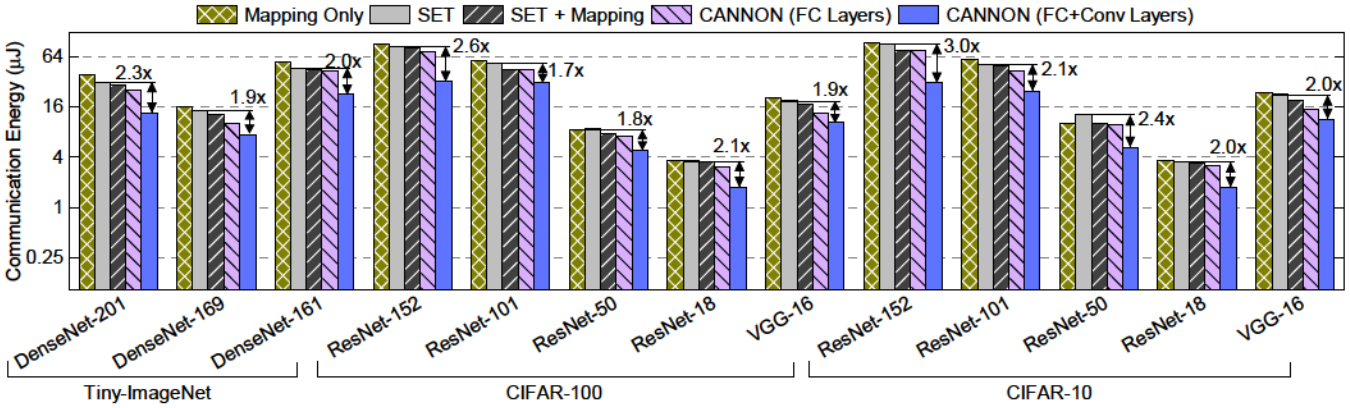


Fig. 9. Comparison of communication **energy** (log scale) across different DNNs and datasets. CANNON consistently improves the communication energy for all network models and datasets.

VGG-16 and MLP Results: We also show the results with VGG-16 network with CIFAR-100 and CIFAR-10 datasets in Figure 8. *CANNON (FC+Conv Layers)* shows a speedup of up to 3.1× for VGG-16 with negligible accuracy impact. Finally, experiments with an MLP on MNIST dataset obtain 2.4× lower communication latency than *SET* (6.6 μs vs 3.7 μs) with similar accuracy. These results are excluded from Figure 8 for readability since the latency for MLP is three orders of magnitude smaller than other networks.

4.4 Energy and EDP Analysis

In this section, we compare the communication energy consumption of CANNON against four different approaches. The details of the communication energy results for DenseNet, ResNet, and VGG-16 models on the Tiny-ImageNet, CIFAR-100, and CIFAR-10 datasets are given in Figure 9. We present detailed energy-delay product (EDP) results in Figure 10.

DenseNet Network Results: The *Mapping Only* approach shows the highest communication energy for DenseNet-201 and DenseNet-161 networks. The *SET + Mapping* approach, where our latency-aware mapping is utilized together with the pruning technique in SET, performs better than the *SET* and *Mapping Only* approaches. In this case, the gain against the *SET* approach shows that our latency-aware mapping reduces energy consumption. Furthermore, we observe a 10%–22% reduction in communication energy consumption with *CANNON (FC Layers)* against *SET + Mapping*. Finally, *CANNON (FC+Conv Layers)* consistently outperforms all other approaches. Specifically, we observe 48%-58% reduction in communication energy compared to *SET*.

The communication latency and energy reduction also result in significant EDP improvements. Since the *Mapping Only* approach does not consider pruning, it has the highest EDP among all techniques for all DenseNet networks on the Tiny-ImageNet dataset. The *SET + Mapping* approach has 7%–20% improvement with respect to the *SET* approach. *CANNON (FC Layers)* further improves the EDP by 9%–39%. Furthermore, *CANNON (FC+Conv Layers)* outperforms all other approaches consistently and shows an EDP improvement of 3.5×–6.2× compared to *SET* (Figure 10).

ResNet Network Results: The ResNet results include four DNNs and two datasets. The *Mapping Only* approach has the highest communication energy among all techniques except for the ResNet-50 model on both datasets. The improvements of *SET + Mapping* over *Mapping Only* are 4%–21% and 5%–22% on CIFAR-100 and CIFAR-10 datasets, respectively. CANNON consumes the least communication energy compared to all other approaches. While *CANNON (FC Layers)* has 1.2×–1.3× improvement over *SET*, *CANNON*

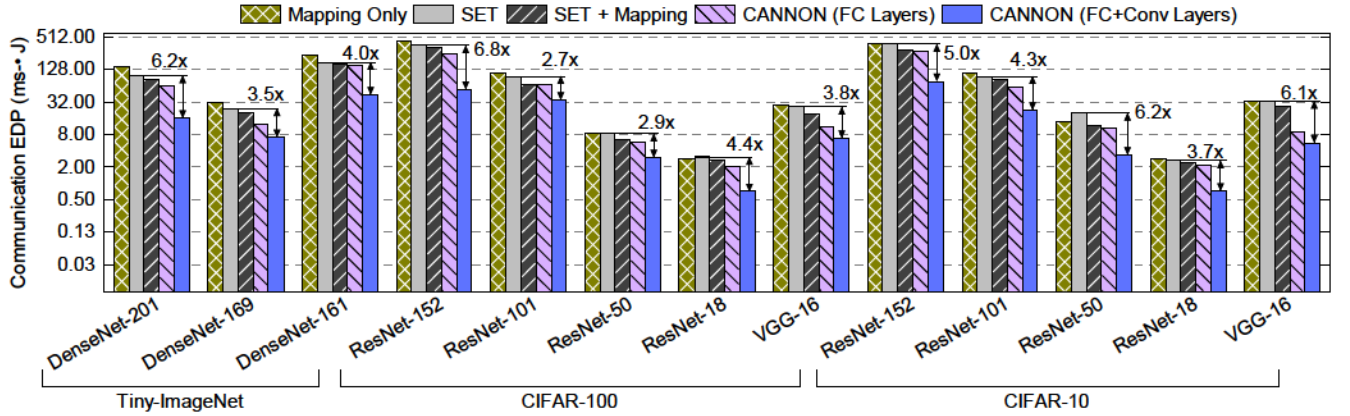


Fig. 10. Comparison of communication **EDP** (log scale) across different DNNs and datasets. As shown, CANNON consistently improves the communication EDP for all network models and datasets.

TABLE 4
Accuracy (%) comparison across different models and datasets. DN and RN denote DenseNet and ResNet, respectively.

Dataset	Tiny-ImageNet (%)			CIFAR-100 (%)					CIFAR-10 (%)					MNIST (%)
Network	DN201	DN169	DN161	RN152	RN101	RN50	RN18	VGG16	RN152	RN101	RN50	RN18	VGG16	MLP
Mapping Only [5]	60.15	60.45	62.04	80.26	79.37	78.52	77.82	75.12	90.74	90.14	89.87	84.16	90.70	98.22
SET [4]	61.40	61.53	62.71	78.98	80.04	79.17	78.01	75.78	90.52	91.10	90.55	88.94	91.37	98.68
CANNON (FC Layers)	60.84	61.22	62.28	78.83	79.54	77.80	77.65	74.81	89.16	90.32	89.77	88.23	90.58	98.34
CANNON (FC+CONV Layers)	59.50	58.74	60.95	78.78	78.06	76.90	76.39	74.58	88.88	88.96	87.98	85.66	89.31	NA
Accuracy Difference from [5]	**-0.65**	**-1.71**	**-1.09**	**-1.48**	**-1.31**	**-1.62**	**-1.43**	**-0.54**	**-1.86**	**-1.18**	**-1.89**	**1.50**	**-1.39**	**0.02**

(FC+Conv Layers) has an improvement of $1.7\times$–$2.6\times$ with respect to *SET* on the CIFAR-100 dataset. On the CIFAR-10 dataset, *CANNON (FC+Conv Layers)* further improves the communication energy up to $3.0\times$ compared to *SET*, as shown in Figure 9.

Similar to energy consumption, CANNON achieves significant improvements in EDP too with respect to all other approaches. The improvement varies between $2.7\times$–$6.8\times$ and $3.7\times$–$6.2\times$ on CIFAR-100 and CIFAR-10 datasets, respectively, when compared to *SET*, as shown in Figure 10.

VGG-16 and MLP Results: We show the energy and EDP results with VGG-16 on CIFAR-100 and CIFAR-10 datasets in Figure 9 and Figure 10, respectively. *CANNON (FC+Conv Layers)* achieves $1.9\times$ and $2.0\times$ communication energy savings with respect to *SET* for VGG-16 on CIFAR-100 and CIFAR-10 datasets, respectively. EDP improvements are even higher with $3.8\times$ and $6.1\times$. Finally, we performed experiments with MLP using MNIST dataset. *CANNON (FC+Conv Layers)* achieves $1.9\times$ lower energy consumption and $4.4\times$ lower EDP than *SET* (105.7 nJ vs 195.8 nJ) without any accuracy impact. MLP results are not shown in Figure 9 and 10 for readability. These savings offer huge advantages for edge devices as they have limited energy budgets.

4.5 Accuracy Analysis

The previous sections show that CANNON significantly improves the hardware performance and energy consumption efficiency compared to the baseline techniques. In this section, we discuss the test accuracy results of CANNON for all DNNs and datasets. All results are shown in Table 4.

DenseNet Network Results: The *Mapping Only* approach [5] using an unpruned DenseNet-201 network model on the Tiny-ImageNet dataset achieves 60.15% test accuracy. As shown in Table 4, *CANNON (FC Layers)* improves the test accuracy by 0.69% with significant hardware efficiency improvements compared to *Mapping Only*, as discussed in previous sections. Moreover, *CANNON (FC Layers)* has 0.56% decrease from the *SET* technique [4]. For DenseNet-169 and DenseNet-161, we also achieve up to 0.77% and 0.24% accuracy improvements over *Mapping Only*, respectively. Table 4 shows that *CANNON (FC Layers)* yields 0.65% decrease from the unpruned network on DenseNet-201. Similarly, we have only 1.71% and 1.09% accuracy loss over the unpruned networks of DenseNet-169 and DenseNet-161, respectively.

ResNet Network Results: *SET* achieves the highest test accuracy in most ResNet-based networks using CIFAR-100 and CIFAR-10 datasets. The *Mapping Only* approach has an accuracy difference in the range of -1.28% to 0.96% compared to the *SET* technique, except for ResNet-18 on CIFAR-10. The *Mapping Only* approach has 4.78% lower accuracy than *SET* for ResNet-18 on CIFAR-10. *CANNON (FC Layers)* has an accuracy difference as minimal as between -1.43% to 0.17% compared to the *Mapping Only* approach on unpruned networks on the CIFAR-100 dataset. The accuracy improvement ranges from -1.58% to 0.18% compared to the *Mapping Only* approach on the CIFAR-10 dataset, except for ResNet-18. For ResNet-18 on CIFAR-10, the accuracy improvement is up to 4.07%. *CANNON (FC+Conv Layers)* shows an accuracy difference between -1.89% to 1.50% compared against the *Mapping Only* approach on both datasets.

We remark that CANNON yields a significant hardware efficiency improvement with less than 2% accuracy loss compared to the unpruned networks. It is then possible to use CANNON with lower pruning ratios to compensate for the accuracy loss.

VGG-16 and MLP Results: As shown in Table 4, CANNON achieves almost the same accuracy as *SET* for MLP. Moreover, it shows 0.12% accuracy improvement over *Mapping Only*. For VGG-16, CANNON shows 0.54% and 1.39% accuracy loss for CIFAR-100 and CIFAR-10, respectively.

4.6 Ablation Study

In this section, we conduct a new ablation study to examine how hardware awareness impacts pruning. We compare two approaches: *CANNON (FC+Conv Layers)* and a communication-agnostic *Baseline (FC+Conv Layers)*. CANNON prunes the network by considering both weight magnitudes and communication costs, whereas *Baseline (FC+Conv Layers)* prunes solely based on weight magnitudes without considering communication costs. Both approaches use the same pruning ratios for fairness.

Table 5 presents that both approaches achieve comparable accuracy. However, *CANNON (FC+Conv Layers)* outperforms *Baseline (FC+Conv Layers)* by up to $1.74\times$ higher performance, with a timing improvement from 0.7ms to 0.41ms.

TABLE 5
CANNON (FC+Conv Layers) comparison with respect to *Baseline (FC+Conv Layers)*. Model and dataset combinations are ResNet-18 on CIFAR-10, VGG-16 on CIFAR-100, and DenseNet-169 on Tiny-ImageNet.

Model	ResNet-18			VGG-16			DenseNet-169		
Metric	Acc. (%)	Lat. (ms)	EDP (msμJ)	Acc. (%)	Lat. (ms)	EDP (msμJ)	Acc. (%)	Lat. (ms)	EDP (msμJ)
Baseline (FC+Conv)	85.42	0.70	2.32	74.88	1.11	19.14	59.21	1.62	22.70
CANNON (FC+Conv)	85.66	0.41	0.72	74.58	0.66	6.85	58.74	0.93	7.04
Improv.	**0.28**	**1.70**$\times$	**3.24**$\times$	**-0.40**	**1.68**$\times$	**2.79**$\times$	**-0.80**	**1.74**$\times$	**3.22**$\times$

TABLE 6
CANNON comparison with respect to lottery ticket hypothesis [6].

Dataset	ResNet-152				ResNet-101				ResNet-50			
Metric	Accuracy (%)	Latency (ms)	Energy (μJ)	EDP (ms-μJ)	Accuracy (%)	Latency (ms)	Energy (μJ)	EDP (ms-μJ)	Accuracy (%)	Latency (ms)	Energy (μJ)	EDP (ms-μJ)
LTH [6]	91.08	3.48	68.09	237.12	91.82	1.31	42.34	55.55	91.57	1.00	9.08	9.08
LTH + Mapping [6]	91.08	3.45	66.87	230.69	91.82	1.23	38.81	47.82	91.57	0.88	8.44	7.42
CANNON	88.88	2.41	31.30	75.38	88.96	0.90	24.76	22.22	87.98	0.62	5.32	3.28
Improvement over LTH [6]	**-2.20%**	**1.45×**	**2.18×**	**3.15×**	**-2.86%**	**1.46×**	**1.71×**	**2.50×**	**-3.59%**	**1.61×**	**1.70×**	**2.77×**

Additionally, *CANNON (FC+Conv Layers)* results in up to a $3.24\times$ reduction in EDP compared to *Baseline (FC+Conv Layers)*. These results highlight the importance of hardware-aware training for achieving optimal performance.

4.7 Comparison to Lottery Ticket Hypothesis

This section compares the accuracy and performance of CANNON with FC and CONV layer pruning against the Lottery Ticket Hypothesis (i.e., LTH) [6] using the official implementation of LTH from the GitHub repository [43]. We use three iterations, each pruning 20% of the network during the training process. In the end, the network of the winning ticket is approximately 49% pruned. We use an equal pruning ratio in CANNON to make the comparison fair. We utilize three networks from the ResNet family with 152, 101, and 50 layers on the CIFAR-10 dataset. We incorporate the mapping method used in [37] (LTH) and our proposed latency-aware mapping (LTH + Mapping) to evaluate the hardware performance of LTH pruning. The accuracy, latency, energy, and EDP results are shown in Table 6. Since our proposed technique prunes a DNN considering hardware performance, CANNON consistently outperforms LTH in all three hardware performance metrics with slightly lower accuracy. The latency improvements with CANNON are $1.45\times$–$1.61\times$ compared against LTH with the mapping method used in [37]. We observe such an improvement due to our proposed latency-aware mapping and hardware-aware dynamic sparse training methodologies, which reduces the communication cost without pruning the significant weights. We also observe that the improvements in energy consumption vary between $1.70\times$ – $2.18\times$. The highest improvements are seen in the energy-delay product (EDP). CANNON achieves up to $3.15\times$ improvement in EDP with respect to the Lottery Ticket Hypothesis with the mapping method used in [37].

As a conclusion, we note that the reduction in network size and the increase in accuracy due to pruning do *not* necessarily translate into better hardware results unless they consider hardware-aware mapping and pruning methods. Overall, CANNON provides excellent results compared to state-of-the-art pruning methods and can work synergistically with any pruning technique.

5 Conclusion

In this paper, we have presented a novel and comprehensive communication-aware sparse DNN optimization framework called CANNON. CANNON first maps the DNN layers to the tiles of the NoC on the target hardware; then, it replaces the fully connected and convolutional layers with sparse graphs. After that, a novel hardware-aware dynamic sparse training technique prunes connections based on the absolute value of their weights and the communication cost on the target hardware. After hardware-aware pruning, we insert new weights based on the communication cost. We maintain the number of parameters throughout the training by pruning and adding the same percentage of weights.

We evaluated the accuracy and latency impact of CANNON extensively on a wide range of DNNs using popular datasets. Our results show that CANNON achieves up to $3.1\times$ lower communication latency and $6.8\times$ lower energy-delay product compared to state-of-the-art pruning approaches [4], [6] and offers substantial energy savings without a significant loss in classification accuracy.

References

[1] C. Gamanayake, L. Jayasinghe, B. K. K. Ng, and C. Yuen, "Cluster Pruning: An Efficient Filter Pruning Method for Edge AI Vision Applications," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 802–816, 2020.

[2] V. Radu, K. Kaszyk, Y. Wen, J. Turner, J. Cano, E. J. Crowley, B. Franke, A. Storkey, and M. O'Boyle, "Performance Aware Convolutional Neural Network Channel Pruning for Embedded GPUs," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 24–34.

[3] S. Yang, W. Chen, X. Zhang, S. He, Y. Yin, and X.-H. Sun, "Auto-prune: Automated DNN Pruning and Mapping for ReRAM-based Accelerator," in *Proceedings of the ACM International Conference on Supercomputing*, 2021, pp. 304–315.

[4] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable Training of Artificial Neural Networks with Adaptive Sparse Connectivity Inspired by Network Science," *Nature communications*, vol. 9, no. 1, pp. 1–12, 2018.

[5] N. Wu, L. Deng, G. Li, and Y. Xie, "Core placement optimization for multi-chip many-core neural network systems with reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 2, pp. 1–27, 2020.

[6] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[7] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A Circuit-level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3067–3080, 2018.

[8] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator," in *IEEE ISPASS*, 2013, pp. 86–96.

[9] 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.

[10] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

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

[12] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015.

[13] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[14] Y. LeCun, "The mnist database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*, 1998.

[15] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[16] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*. IEEE, 1993, pp. 293–299.

[17] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[18] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2498–2507.

[19] F. Karimzadeh, N. Cao, B. Crafton, J. Romberg, and A. Raychowdhury, "Hardware-aware pruning of dnns using lfsr-generated pseudo-random indices," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[20] C. Chu, Y. Wang, Y. Zhao, X. Ma, S. Ye, Y. Hong, X. Liang, Y. Han, and L. Jiang, "Pim-prune: Fine-grain dcnn pruning for crossbar-based process-in-memory architecture," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[21] J. Meng, L. Yang, X. Peng, S. Yu, D. Fan, and J.-S. Seo, "Structured pruning of rram crossbars for efficient in-memory computing acceleration of deep neural networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1576–1580, 2021.

[22] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[23] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 1–18, 2017.

[24] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 806–814.

[25] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 548–560, 2017.

[26] W. Romaszkan, T. Li, and P. Gupta, "3pxnet: Pruned-permuted-packed xnor networks for edge machine learning," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 19, no. 1, pp. 1–23, 2020.

[27] G. Krishnan, X. Du, and Y. Cao, "Structural pruning in deep neural networks: A small-world approach," *arXiv preprint arXiv:1911.04453*, 2019.

[28] G. Yuan, P. Behnam, Y. Cai, A. Shafiee, J. Fu, Z. Liao, Z. Li, X. Ma, J. Deng, J. Wang, M. Bojnordi, Y. Wang, and C. Ding, "Tinyadc: Peripheral Circuit-aware Weight Pruning Framework for Mixed-signal DNN Accelerators," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 926–931.

[29] J. Lin, Z. Zhu, Y. Wang, and Y. Xie, "Learning the sparsity for reram: Mapping and pruning sparse neural network for reram based accelerator," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 639–644.

[30] L. Liang, L. Deng, Y. Zeng, X. Hu, Y. Ji, X. Ma, G. Li, and Y. Xie, "Crossbar-aware neural network pruning," *IEEE Access*, vol. 6, pp. 58 324–58 337, 2018.

[31] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 236–249.

[32] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson, "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," in *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, 2009, pp. 233–244.

[33] Y. Saad, "Sparskit: a basic tool kit for sparse matrix computations-version 2," 1994.

[34] S. Smith and G. Karypis, "Tensor-matrix products with a compressed sparse tensor," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, 2015, pp. 1–7.

[35] G. Krishnan, S. K. Mandal, C. Chakrabarti, J.-s. Seo, U. Y. Ogras, and Y. Cao, "Interconnect-aware Area and Energy Optimization for In-Memory Acceleration of DNNs," *IEEE Design & Test*, vol. 37, no. 6, pp. 79–87.

[36] S. K. Mandal, G. Krishnan, C. Chakrabarti, J.-S. Seo, Y. Cao, and U. Y. Ogras, "A Latency-optimized Reconfigurable NoC for In-memory Acceleration of DNNs," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 362–375, 2020.

[37] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with in-situ Analog Arithmetic in Crossbars," *ACM/IEEE ISCA*, 2016.

[38] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A Pipelined Reram-based Accelerator for Deep Learning," in *IEEE HPCA*, 2017, pp. 541–552.

[39] E. Paul and R. Alfréd, "On random graphs i," *Publicationes Mathematicae (Debrecen)*, vol. 6, pp. 290–297, 1959.

[40] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[41] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

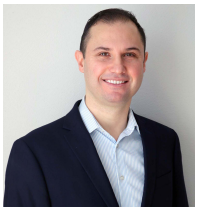
[42] X. Qiao, X. Cao, H. Yang, L. Song, and H. Li, "Atomlayer: A universal reram-based cnn accelerator with atomic layer computation," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[43] J. Frankle, "Openlth: A framework for lottery tickets and beyond," https://github.com/facebookresearch/open_lth, 2020.

A. Alper Goksoy (Graduate Student Member, IEEE) received his B.S. degree in Electrical and Electronics Engineering from Bogazici University, Istanbul, Turkey. He is currently pursuing his Ph.D. in Electrical Engineering at University of Wisconsin-Madison, USA. His research interests include the design of AI hardware, task scheduling for heterogeneous, domain-specific SoCs, and graph neural networks. He received Richard Newton Young Fellowship at DAC 2019, and Young Fellowship at DAC 2021.

Guihong Li (Graduate Student Member, IEEE) received the B.S degrees from the Beijing University of Posts and Telecommunications, Beijing, China, in 2018. He is currently pursuing his Ph.D. in Electrical and Computer Engineering at The University of Texas at Austin, USA. His research interest includes Neural Architecture Search, hardware-software co-design for EdgeAI system optimization. He received numerous awards including best paper nomination from ISWC 2022.

Sumit K. Mandal (Member, IEEE) received the dual (B.Tech.+M.Tech.) degrees from the Indian Institute of Technology, Kharagpur, India, in 2015. He completed his Ph.D. in Electrical Engineering with University of Wisconsin-Madison, USA in 2022. He is currently an Assistant Professor in the Department of Computer Science and Automation at Indian Institute of Science, Bangalore. His research interest includes analysis and design of NoC architecture, AI hardware and power management of multicore processors. He received numerous awards including best paper award from ACM TODAES in 2021.



Umit Y. Ogras received his Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University, Pittsburgh, PA, in 2007. He is currently an Associate Professor in the Dept. of Electrical and Computer Engineering at the University of Wisconsin-Madison. His research interests include embedded systems, heterogeneous systems-on-chip, low-power VLSI, wearable computing, and flexible hybrid electronics. Dr. Ogras received DARPA Director's Fellowship Award (2020), DARPA Young Faculty Award (2018), NSF CAREER Award (2017), Intel Strategic CAD Lab Research Award (2013), and best paper awards at 2019 CASES, 2017 CODES+ISSS, 2012 IEEE Transactions on CAD, and 2011 IEEE VLSI Transactions.

Radu Marculescu is the Laura Jennings Turner Chair in Engineering and Professor in the Electrical and Computer Engineering department at The University of Texas at Austin. He received his Ph.D. in Electrical Engineering from the University of Southern California in 1998. Radu's current research focuses on developing ML/AI methods and tools for modeling and optimization of embedded systems, cyber-physical systems, and social networks.