This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3064466, IEEE Transactions on Mobile Computing

Radio Frequency Fingerprinting on the Edge

Tong Jian, Yifan Gong, Zheng Zhan, Runbin Shi, Nasim Soltani, Zifeng Wang,
Jennifer Dy, Kaushik Chowdhury, Yanzhi Wang, and Stratis Ioannidis

***Abstract*—Deep learning methods have been very successful at radio frequency fingerprinting tasks, predicting the identity of transmitting devices with high accuracy. We study radio frequency fingerprinting deployments at resource-constrained edge devices. We use structured pruning to jointly train and sparsify neural networks tailored to edge hardware implementations. We compress convolutional layers by a $27.2\times$ factor while incurring a negligible prediction accuracy decrease (less than 1%). We demonstrate the efficacy of our approach over multiple edge hardware platforms, including a Samsung Gallaxy S10 phone and a Xilinx-ZCU104 FPGA. Our method yields significant inference speedups, $11.5\times$ on the FPGA and $3\times$ on the smartphone, as well as high efficiency: the FPGA processing time is $17\times$ smaller than in a V100 GPU. To the best of our knowledge, we are the first to explore the possibility of compressing networks for radio frequency fingerprinting; as such, our experiments can be seen as a means of characterizing the informational capacity associated with this specific learning task.**

I. Introduction

With billions of pervasively deployed and connected devices, the oncoming IoT revolution will create a new paradigm of sensing and actuation at the network edge. While certain types of environmental, industrial and human-activity centric sensing require relaying massive amounts of data from the field sensor to the remote cloud, there are many scenarios that can benefit from preliminary analysis and inference at the edge itself [1]. An important but often overlooked aspect of sensing is authenticating the identity of the sensors that generate the field data by other peer devices. If this step can occur at the edge, it may prevent power-constrained IoT devices from incurring significant processing cycles and communication overheads by restricting information flow towards the cloud, where more complex and upper-layer security mechanisms are available.

We focus on a specific example of radio frequency (RF) fingerprinting to authenticate a transmitter by an edge aggregation gateway or a trusted IoT device responsible for further relaying of the sensed information. The concept of RF fingerprinting, like its traditional human counterpart, rests on the assumption that there are unique discriminative features for an individual device, which can be used to identify it from a pool of similar devices. Process variations in the manufacturing supply chains impose subtle changes in the tolerances and properties of the electrical components that compose the front-end of a wireless transmitter [2]. These variations, often occurring at multiple processing blocks of the transmitter chain, shift the operating point of the transmitted signal in the in-phase/quadrature (IQ) representation space [3].

The authors are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, USA.

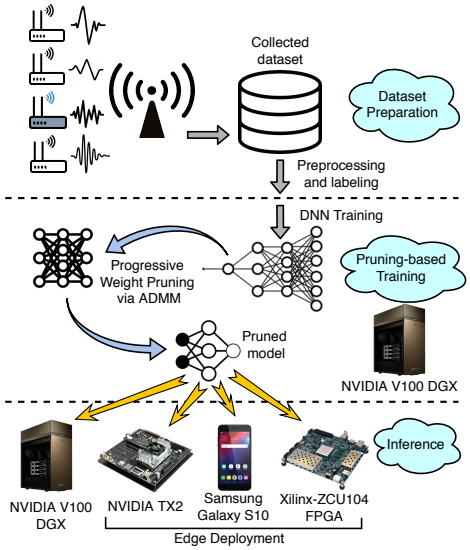


Fig. 1. An overview of our edge RF fingerprinting framework. We first preprocess an RF fingerprinting dataset and train a CNN architecture. We then use a progressive weight pruning algorithm to jointly re-train and sparsify the CNN. Finally, we execute the pruned model on different hardware platforms to assess the computation and energy efficiency improvement at inference time.

Some examples of such variations are IQ imbalance, DC offset, non linear power amplifier gain, among others. Since the specific discriminative function is unknown for a specific transmitter, it is difficult to create analytical models tuned for a particular device. Deep learning for RF fingerprinting, in which a large dataset of signals collected from the transmitter is used to train a convolutional neural network (CNN), has been very successful [4]–[7] at extracting discriminative fingerprints, indeed enabling authentication at the physical layer.

RF fingerprinting using neural networks typically performed offline [8], [9]: data is collected and moved to a server where the inference is performed using a powerful GPU. However, moving the data to a remote server is not efficient for online RF fingerprinting applications. In recent years, edge computing has developed solutions for bringing computations from remote servers and the cloud closer to the sensors, cameras, radio receivers, etc., in order to eliminate data transfer bottlenecks, reduce latency, and accelerate inference. Our objective is to train a deep CNN in the central cloud, and subsequently deploy it on a low power edge device. Given that large datasets, capable GPU clusters and massive power availability are required for training, we leverage the cloud for this training task. However, the trained model is to be disseminated to the edge devices, to enable inference on site. Performing fingerprinting at the physical layer on the edge device avoids the delivery of GBs of IQ samples as packet payload over

a multi hop wireless link to the cloud, which is infeasible. In addition to eliminating the data transfer bottleneck, edge deployment of the models also opens up the opportunity for low-latency inference directly on the edge device. In turn, RF fingerprinting on the edge gives rise to challenges, precisely due to the practical reality of limited processing power and memory available in edge devices (compared to the GPU-enabled cloud).

As a result, low-latency, efficient fingerprinting on the edge requires a careful design of the neural network so that it (i) attains the high prediction accuracy afforded by CNN architectures, while (ii) satisfying inherent hardware limitations. This necessitates generating compressed, parsimonious network representations, that fit (and can be executed efficiently on) dedicated hardware, without a loss of accuracy due to parsimony. In addition, efficiency necessitates a careful software-hardware co-design: neural networks should be compressed, if possible, in ways that exploit and are tailored to the hardware characteristics of edge devices.

We make the following contributions:

- We use *structured pruning* [10], to produce highly compressed versions of our RF fingerprinting neural networks that are tailored to fast inference at the edge. In particular, focusing on the (computatially intensive) convolutional layers, we reduce weights by a $27.2\times$ scaling factor while incurring a negligible prediction accuracy drop (0.54% on WiFi and 0.44% on ADS-B transmissions).
- We leverage an Alternating Direction Method of Multipliers (ADMM) method for pruning the network *during training* [11], as well as a progressive pruning technique [12], that increases model parsimony gradually. Compared to training a parsimonious model directly (i.e. without pruning), these combined methods lead to a 9.20% accuracy improvement. We make our code publicly available[1] to accelerate community contributions in this exciting topic.
- Our structured pruning approach is designed in a way that leverages edge hardware characteristics, in particular by enabling parallelizability despite its parsimony. We provide implementations on multiple edge device hardware, including a Samsung GalaxyS10 cell phone and a Xilinx-ZCU104 FPGA. Our FPGA implementation in particular is custom-designed for our parsimonious fingerprinting architecture.
- We extensively evaluate the performance of our approach. Pruning yields significant speedups ($\sim$11.5$\times$ on the FPGA, and $\sim 3\times$ on the smartphone). Moreover, FPGA hardware achieves the best efficiency, which is $17\times$, $18\times$, and $14\times$ better than the V100, TX2, and smartphone-GPU, respectively.

Our approach is summarized in Fig. 1. To the best of our knowledge, our work is the first comprehensive investigation of compressing an RF fingerprinting CNN architecture through pruning; as such, our experiments can be seen as a means of characterizing the neural network "informational capacity" required to perform this learning task. Our work is also the first to provide as systems-level analysis of the implementation of such a pruning architecture on edge devices.

The remainder of this paper is structured as follows. We review related work in Section II. In Section III, we describe our dataset and RF fingerprinting CNN architecture in detail. In Section IV, we describe the progressive weight pruning algorithm we use to jointly re-train and sparsify the CNN. Next, we present three edge hardware implementations in Section V and our experimental evaluation in Section VI. Finally, we conclude in Section VII.

II. Related Work

Limitations of RF Fingerprinting Techniques. RF fingerprinting enables device identification by learning unchanging, hardware-based characteristics of the transmitter. A transmitter's processing chain introduces unchangeable and non-linear artifacts in the transmitted signals, which are presented in the in-phase (I) and quadrature components (Q) sequences. These imperfections include information about IQ imbalance, phase noise, and carrier frequency offset, all of which serve as a unique identity, i.e. fingerprint, to the transmitter [13]–[16]. Many existing works extract such fingerprints through complex and protocol-specific feature extraction techniques [17], [18]. These techniques require domain knowledge, such as packet frame structures, channel bandwidths, modulation choices, and coding schemes. This reduces robustness and cross-protocol compatibility: these fingerprinting algorithms require a manual redesign when exposed to new datasets or protocols. To avoid these drawbacks, recent works explore machine learning techniques that are robust to communication protocol changes, and aim to automatically learn hardware-specific features without hand-engineering via neural networks [19]–[22].

The success of CNNs in a variety of domains [23]–[25] has led to considerable recent activity in their application to RF fingerprinting. More broadly, there has been extensive prior research [8], [20], [21], [26]–[31] has extensively explored the efficacy and performance of different classifiers applied to RF fingerprinting. In particular, Riyaz *et al.* [20] and Ali *et al.* [29] explored several popular architectures on RF fingerprinting and compared them with traditional ML techniques (e.g., SVM and logistic regression). Deep CNNs consistently outperformed these classifiers over several datasets [20]. Jian *et al.* show that CNNs are resilient to MAC spoofing [22]. Jian *et al.* [8] compared several CNN architectures such as AlexNet-1D [23], GoogleNet-1D [24], VGG16-1D [32] and ResNet50-1D [25] over multiple datasets capturing different environmental scenarios, including the impact of number of classes, the channel variation, SNR, and training dataset size. We adopt the ResNet50-1D architecture from Jian *et al.* [8] for our exploration on pruning as it was shown to be the best-in-class in this earlier work.

Unfortunately, CNN-based RF fingerprinting comes at significant computational costs induced by the depth of these architectures. Close to our work, Soltani *et al.* [33] use Android smartphones with TensorFlow Lite for RF fingerprinting, but do not exploit hardware-software co-design. To the best of our

[1] https://github.com/neu-spiral/RFonEdge

knowledge, we are the first in the field of RF fingerprinting to explore the possibility of (i) using pruned, parameter-constrained models (ii) on low-cost and resource-limited devices.

Model Compression. The growth of the number of parameters in modern CNNs [32], [34], [35] impedes the deployment of models on devices that have limited on-chip resources. Weight pruning is one of the major compression techniques to reduce model size. Early works of unstructured weight pruning achieve a considerable reduction in the number of weights of representative CNNs with minor accuracy loss [10], [11], [36]–[38]. However, as discussed in Sec. IV, the resulting irregular, sparse matrices are not friendly to hardware acceleration during inference.

To overcome these limitations, later works [39]–[46] propose *structured pruning*, by incorporating regularity into weight pruning. Structured pruning can be further categorized into filter pruning [39], [47] and column pruning [48], [49]. Filter pruning removes whole filters, and column pruning (filter shape pruning) prunes weights for all filters at the same locations in a layer. Channel pruning [39], which prunes channels completely from the feature map, is essentially equivalent to filter pruning, as pruning filters in a layer removes the corresponding channels of the next layer. As convolutions in CNNs is commonly transformed in hardware into GEneral Matrix Multiplications (GEMMs) by converting weight tensors and feature map tensors to matrices, structured pruning methods can directly reduce the dimensions of weight matrices while maintaining a full matrix format, thus facilitating hardware implementations; we elaborate on this further in Sec. IV. Nevertheless, the success of pruning, structured or otherwise, depends directly on the capacity of the resulting network, and the complexity of the inference task at hand. To the best of our knowledge, we are the first to explore the possibility of compressing networks for RF fingerprinting; as such, our experiments can be seen as a means of characterizing the informational capacity associated with this specific learning task.

Besides weight pruning, there also exist other model compression approaches including low-rank factorization [50], [51], transferred/compact convolutional filters [52], [53], and knowledge distillation [54]–[56]. Low-rank factorization methods leverage matrix/tensor decomposition to decompose an original large model to a compact model with more lightweight layers. However, these methods involves computationally-expensive decomposition operations and can only be applied layer by layer. Methods based on transferred/compact convolutional filters design special structural convolutional filters to reduce storage and computation cost. As for knowledge distillation, they compress deep and wide networks into shallower ones that can mimic the function learned by the complex model. The main idea is to transfer knowledge from a large teacher model into a smaller student model through learning the class distributions output. Compared with other approaches, weight pruning (i) has a widely used application scenarios, (ii) can be applied to different settings with high performance, and (iii) can be easily combined with other compression approaches for further improvements and speedup. Thus we mainly focus on weight pruning as model compression approach in this paper.

Inference Acceleration via Parallel Hardware. Even under compression, CNN inference still incurs a tremendous computational workload, and therefore a long latency and high energy cost, that impedes the real-time implementation over edge devices. Parallel hardware, including signal instruction multiple data (SIMD) processors, graphic processing unit (GPUs) and field-programmable gate arrays (FPGAs), are accessible in modern devices, which can be leveraged to speed up inference. Previous works on such hardware implementations study the parallel acceleration of 2D convolutional layers [57], [58]; the 1D convolutions we employ in our architecture (see Sec. III) have not received any attention. Our systematic design for RF fingerprinting acceleration on FPGAs (Sec. V) addresses this. Departing from the existing CNN accelerators [57], [58], we design the architecture with dataflow optimization for 1D convolution and compressed weight streaming for our structured pruning approach, illustrating that a tailored design can attain high efficiency.

Accelerating CNN inference on edge devices has received considerable attention in recent years, and has lead to frameworks such as MCDNN [59], DeepMon [60], TFLite [61], TVM [62], and Alibaba Mobile Neural Network [63]. However, most of these prior works do not explore optimization opportunities such as computation and memory foot print reductions offered by model compression techniques. Efforts on accelerating CNN inference by model compression include Liu et al. [64], DeftNN [65], SCNN [66], AdaDeep [67]. These prior works either (i) attain a worse trade off pruning rate and accuracy than the one we report here [67], (ii) do not target edge devices [64], or (iii) require new hardware [65], [66]. Finally, TensorRT [68] has been used to perform fast and low-power inference on the edge GPU NVIDIA Jetson TX2 in a variety of contexts [69]–[71]. A quintessential example includes mounting the light-weighted TX2 board on drones for fast object detection [70], [71]. To the best of our knowledge, we are the first to profile TX2 implementations of RF fingerprinting via Pytorch and TensorRT.

III. RF Fingerprinting Architecture

We begin by describing the architecture we use for RF fingerprinting. This is to be trained over a dataset of wireless transmissions, and then deployed on edge devices. In this section, we give an overview of the dataset, pre-processing steps, and the CNN architecture (i.e., ResNet50-1D) that we use for training and model pruning.

A. RF Fingerprinting Dataset

We consider a dataset that comprises wireless transmissions collected from two different wireless standards: (i) commercial off-the-shelf (COTS) 802.11b/g/n WiFi and (ii) the Automatic Dependent Surveillance-Broadcast (ADS-B) standard used in airplane status updates. Our WiFi dataset contains 500 devices in total and 273 transmissions for each device collected in a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3064466, IEEE Transactions on Mobile Computing

single day. A spectrum analyzer is used to record each transmission, operating at a 2.4 GHz center frequency with sampling rate 200 MS/s. Each recording consists of 15979 IQ samples, on average. The ADS-B dataset contains 50 devices and 273 transmissions for each device collected under high SNR (ranging from 15.3 to 5.1dB). Each transmission is recorded at a 1.09 GHz center frequency with sampling rate 100 MS/s. Each recording file, besides IQ samples, contains metadata such as the device ID, the transmission frequency, and the protocol used. We refer to the recordings of both WiFi and ADS-B protocols as *transmissions*, for convenience. Hence, each transmission is a variable-length vector of (two-dimensional) IQ samples. Additional details regarding the dataset statistics are provided in Section VI-A (see Table I).

B. Data Pre-Processing

Following Jian *et al.* [8], we perform three pre-processing steps: band filtering, partial equalization, and slicing. The former two are only applied to WiFi transmissions, while the latter is applied to both WiFi and ADS-B transmissions.

Band Filtering. WiFi transmissions are collected in a multi-device environment, i.e., multiple devices transmit signals on multiple bands simultaneously. To remove signals generated out of band and extract clear transmissions made by a specific device, we apply band filtering using the central frequency in the metadata. We perform band filtering on all WiFi transmissions in our datasets and refer the output, the "filtered" transmissions, as *raw WiFi* data. For the ADS-B dataset, we do not perform band filtering, because each transmission happens in isolation without interference.

Partial Equalization. Partial equalization [8], [21] removes channel effects from raw IQ samples while maintaining device-specific imperfections. Specifically, we first estimate and compensate the carrier frequency offsets, then estimate channel using transmission pilot, and reapply the offsets to obtain the final sequence of equalized transmission. Formally, let $\mathbf{y} \in \mathbb{C}^{\mathbf{L}}$ denote the sequence of I and Q components. We use its preamble to estimate and compensate the carrier frequency offsets as follows,

$$\begin{aligned} \gamma &= f_c(\mathbf{y}_{\texttt{preamble}}) \\ \hat{y}[t] &= y[t] \cdot e^{-jt\gamma}, \quad t = 0, 1, 2, \dots \end{aligned} \tag{1}$$

where f_c is the estimation function of carrier frequency offsets [72]. Let $\hat{\mathbf{Y}}$ be the signal after Fast Fourier Transform (FFT) from $\hat{y}[t]$. We estimate the channel using its transmission pilot and do equalization via:

$$\begin{aligned} H_k &= f_e(\hat{\mathbf{Y}}_{\texttt{pilot}}), \quad k = 0, 1, \dots, 63 \\ X_k &= \frac{\hat{Y}_k}{H_k}, \quad k = 0, 1, \dots, 63 \end{aligned} \tag{2}$$

where f_e is the channel estimation function [73] and X_k is the equalized data. Finally, we reapply the offsets via:

$$\begin{aligned} x[t] &= \text{IFFT}(X_k) \\ x[t] &= x[t] \cdot e^{jt\gamma}, t = 0, 1, 2, \dots \end{aligned} \tag{3}$$

where IFFT$(\cdot)$ denotes the inverse FFT. We perform partial equalization on WiFi transmissions and refer to them as *equalized*, as opposed to raw WiFi transmissions. We do not perform partial equalization on the ADS-B dataset. This is because, in our ADS-B dataset, channel conditions do not have prominent effects.

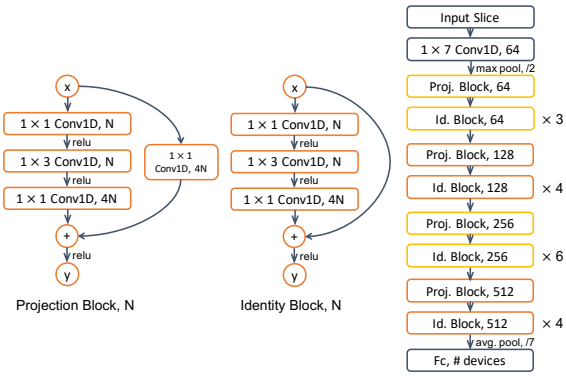


Fig. 2. Our ResNet50-1D architecture contains 49 CLs and 1 fully-connected layer. Compared to the original 2D version, we build ResNet50-1D by only changing the filter width to 1×1 or 1×3 accordingly to match the dimension of IQ slices.

C. CNN Architectures

Among popular deep and large size CNN models [23], [24], [74], ResNet [25], as the Winner of ILSVRC 2015 in image classification, has proved to be remarkably successful across domains. In particular, its extended 1D version, i.e., ResNet50-1D, has shown state-of-art performance in RF fingerprinting domain [8], [9]. In this work, we use ResNet50-1D as the basic architecture for general training and pruning.

As shown in Fig. 2, ResNet50-1D contains 49 convolutional layers (CLs) and 1 fully-connected layer. Every 3 CLs are grouped into a block by an identity mapping shortcut or a projection shortcut, both of which carry forward the input of the block to its output. Shortcuts mitigate the vanishing gradient effect, which is the main cause of accuracy loss in deep architectures. We adapt the original 2D version of convolutional filters to our RF inputs as follows: the filter width of ResNet50-1D is set to 1×1 or 1×3 accordingly, to match the dimension of IQ slices described in the next section. Thus, each kernel learns a variation in time over the I and Q dimension jointly.

D. CNNs for Variable-length Sequences

Slicing. Recall that each WiFi or ADS-B transmission is a variable-length sequence of I and Q components. In order to train and evaluate the (fixed input size) CNNs with these transmissions, we follow the work of Sankhe *et al.* [21] and Jian *et al.* [22]. These works use a sliding window approach to cut transmissions into fixed length slices, which the CNN can ingest. More specifically, a desired slice size l is defined first. Given an entire transmission k of length L_k, we generate $L_k - l + 1$ slices as shown in Fig. 3. We only use a random fraction of slices to train a classifier. A parameter $\kappa \in [1, l]$ governs the number of slices that each IQ sample of transmission k participates in during a training epoch. Slicing and randomization (i) extend our CNN to variable length transmissions, (ii) enhance shift-invariance [22], and (iii) reduce computation costs during training.

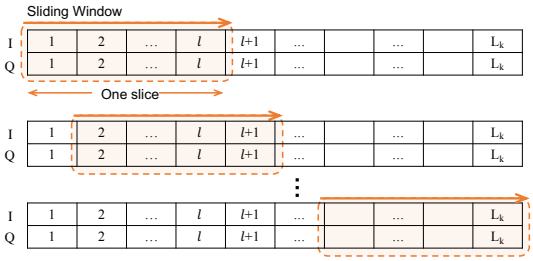


Fig. 3. An illustration slice generation from an arbitrary-length transmission. Given a transmission of length L_k, we create $L_k - l + 1$ slices by sliding a window of size l over the IQ sequence with stride 1. This leads to inputs of a fixed length, and enhances shift invariance.

Transmission Prediction. We describe now how to use the CNN trained over slices to detect the device of the entire transmission. Given a transmission, we slide a window across the entire transmission with a designed stride (as shown in Fig. 3) and evaluate our model on each slice. To predict the device that generated the entire transmission, we sum the probability predictions of slices generated from this specific transmission and declare the target device for which the entry is maximized [8]. Formally, let n_k indicate the number of slices generated from transmission k, and p_{ij} be the model prediction probability that slice j was transmitted by device i. The prediction of transmission k is calculated via $\hat{i} = \arg\max_i \sum_{j=1}^{n_k} p_{ij}$. In practice, during testing, we use strides different than 1, treating this as a hyperparameter.

E. Discussion

In our exploration of pruning, we assume the standard classification setting, whereby class (device) labels are identical in the training and test set. Moreover, our focus is on accelarating *inference*, not training: a trained network is to be compressed so that detection of devices in the training set can be done efficiently at dedicated hardware. In real-life scenarios, the set of devices/classes may vary, and previously unseen devices may need to be detected as such [75]–[77]; alternatively, the classifier may also need to be re-trained to handle new classes [78]–[81]. Addressing such questions while pruning requires a departure from the setting we study here; we discuss extensions in Section VII.

IV. Model Pruning

In this section, we describe the model pruning algorithm we use to achieve our goal of computation reduction and energy efficiency improvement at inference. We prune the model only on convolutional layers (CLs); this is a common practice (see, e.g., [11]). This is because CLs are repeatedly applied during inference; as a result, they contribute the majority of the computational load in state-of-the-art CNNs.

A. Problem Formulation

We use the following notation throughout the paper. For an N-layer CNN, we use the subscript $i \in \{1, \ldots, N\}$ to indicate layers. The 1D convolutional layer in our setting is represented by a three-dimensional weight tensor, with each dimension $p_i, q_i, r_i \in \mathbb{N}$ representing the number of filters, channels, and filter widths, respectively, in each layer i. GEneral Matrix Multiplication operations (GEMMs) on accelerators operate over a reshaped tensor of two dimensions $P_i = p_i$ and $Q_i = q_i \times r_i$. We thus represent each layer via the (reshaped) matrix $\boldsymbol{W}_i \in \mathbb{R}^{P_i \times Q_i}$ and the bias vector $\boldsymbol{b}_i \in \mathbb{R}^{P_i}$. We also define $\boldsymbol{W} := \{\boldsymbol{W}_i\}_{i=1}^N$ and $\boldsymbol{b} := \{\boldsymbol{b}_i\}_{i=1}^N$ as the set of all weights and biases in the CNN. We denote the loss of the CNN under dataset $\mathcal{D}$ by $f(\boldsymbol{W}, \boldsymbol{b}; \mathcal{D})$.

Our objective is to prune non-important weights while preserving the accuracy of the pruned model. Therefore, we minimize the loss function subject to constraints specifying sparsity requirements. More specifically, the weight pruning problem can be formulated as:

$$\begin{aligned} \underset{\boldsymbol{W},\boldsymbol{b}}{\text{Minimize:}} \quad & f(\boldsymbol{W}, \boldsymbol{b}; \mathcal{D}), \\ \text{subject to} \quad & \boldsymbol{W}_i \in S_i, \quad i = 1, \cdots, N, \end{aligned} \tag{4}$$

where $S_i \subseteq \mathbb{R}^{P_i \times Q_i}$ is a weight sparsity constraint set applied to layer i. A broad variety of constraints can be expressed through sets S_i. For example, in unstructured pruning, $S_i = \{\boldsymbol{W}_i \mid \|\boldsymbol{W}_i\|_0 \leq \alpha_i\}$,where $\|\cdot\|_0$ is the size of $\boldsymbol{W}_i$'s support (i.e., the number of non-zero elements), and $\alpha_i \in \mathbb{N}$ is a constant. This constraint indicates that the number of non-zero elements of $\boldsymbol{W}_i$ should not exceed α_i. However, the resulting model is hard to implement efficiently on an edge device, as it requires keeping track of arbitrarily located indices [10], [11], [36]–[38]. This leads to performance degradation when implemented over accelerators [42], [82].

To address this, we follow a *structured* pruning approach, whereby sets S_i are limited *dimension-wise*. Formally, given matrix $\boldsymbol{W}_i$, let $[\boldsymbol{W_i}]_{p,:} \in \mathbb{R}^{Q_i}$, $[\boldsymbol{W_i}]_{:,q} \in \mathbb{R}^{P_i}$ be the p-th row and q-th column of $\boldsymbol{W}_i$, respectively. Moreover, for ϕ a boolean predicate, let $\mathbb{1}_\phi$ to be 1 if ϕ is true, and 0 otherwise. In structured filter pruning, for sparsity parameter $\alpha_i \in \mathbb{N}$, the set S_i is defined as:

$$S_i = \left\{\boldsymbol{W}_i \mid \left(\textstyle\sum_{p=1}^{P_i} \mathbb{1}_{([\boldsymbol{W}_i]_{p,:} \neq \boldsymbol{0})}\right) \leq \alpha_i\right\}. \tag{5}$$

In other words, this constraint enforces that the number of non-zero rows (filters) on the i-th layer does not exceed α_i. A similar constraint can be placed on columns; that is, for sparsity parameter $\alpha_i \in \mathbb{N}$, the corresponding column sparsity constraint is:

$$S_i = \left\{\boldsymbol{W}_i \mid \left(\textstyle\sum_{q=1}^{Q_i} \mathbb{1}_{([\boldsymbol{W}_i]_{:,q} \neq \boldsymbol{0})}\right) \leq \alpha_i\right\}. \tag{6}$$

Intuitively, this enforces that the number of non-zero columns in $\boldsymbol{W}_i$ does not exceed α_i. Both filter and column sparsity constraints are more "hardware friendly" compared to unstrucured pruning [39], [42]: they reduce computations, while still allowing for the benefit of parallelization over the non-zero dimension. We discuss our custom edge hardware implementations that exploit this in Section V.

B. Model Pruning Using ADMM

Problem (4) is non-convex with combinatorial constraints, thus cannot be solved using stochastic gradient descent methods as in standard CNN training. To deal with this, we leverage the Alternating Direction Method of Multipliers (ADMM) to

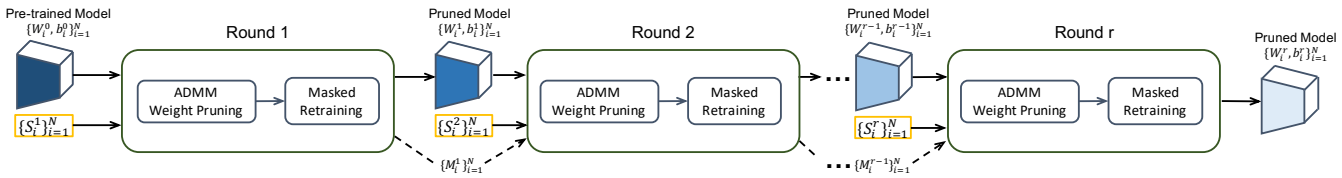


Fig. 4. An illustration of progressive pruning. We prune the network over multiple rounds. In each round, we execute the ADMM pruning algorithm and a masked retraining process to the pre-trained model. The input to each round r is a pre-trained neural network as well as a constraint set $\{S_i^r\}_{i=1}^N$. As the number of round increases, stronger constraints are provided and added to the model. The output of the round r will be a pruned model, consisting of the weight matrix $\mathbf{W}^r$ satisfies the constraint such that $\mathbf{W}_i^r \in S_i^r$. Besides the pruned model, each round could output a mask $\{\boldsymbol{M}^r\}_{i=1}^N$ that contains binary-valued elements with value 0 corresponding to the pruned weights and value 1 corresponding to the remaining weights. We could then prune the model additionally conditioning on this passed mask.

decompose the original problem into two subproblems that can be solved efficiently and separately [10], [11], [83]. We begin by rewriting problem (4) in the ADMM form by introducing auxiliary variables $\boldsymbol{Z}_i$:

$$\begin{aligned} &\underset{\boldsymbol{W},\boldsymbol{b}}{\text{Minimize:}} \quad f(\boldsymbol{W},\boldsymbol{b};\mathcal{D}) + \textstyle\sum_{i=1}^N g_i(\boldsymbol{Z}_i), \\ &\text{subject to} \quad \boldsymbol{W}_i = \boldsymbol{Z}_i, \quad i = 1,\cdots,N, \end{aligned} \tag{7}$$

where $g_i(\cdot)$ is the indicator of set S_i, defined as:

$$g(\boldsymbol{Z}_i) = \begin{cases} 0 & \text{if } \boldsymbol{Z}_i \in S_i, \\ +\infty & \text{otherwise.} \end{cases} \tag{8}$$

The augmented Lagrangian of problem (7) is defined as [83]:

$$\begin{aligned} &\mathcal{L}(\boldsymbol{W},\boldsymbol{b},\boldsymbol{Z},\boldsymbol{U}) = f\big(\boldsymbol{W},\boldsymbol{b};\mathcal{D}\big) + \textstyle\sum_{i=1}^N g_i(Z_i) + \\ &+ \textstyle\sum_{i=1}^N \rho_i \,\mathrm{trace}(\boldsymbol{U}_i^\top(\boldsymbol{W}_i - \boldsymbol{Z}_i)) + \sum_{i=1}^N \frac{\rho_i}{2}\|\boldsymbol{W}_i - \boldsymbol{Z}_i\|_F^2, \end{aligned} \tag{9}$$

where ρ_i is a penalty value and $\mathbf{U}_i \in \mathbb{R}^{P_i \times Q_i}$ is a dual variable, rescaled by ρ_i.

The ADMM algorithm proceeds by repeating the following iterative optimization process until convergence. At the k-th iteration, the steps are given by

$$\boldsymbol{W}^{(k)}, \boldsymbol{b}^{(k)} := \underset{\boldsymbol{W},\boldsymbol{b}}{\arg\min}\, \mathcal{L}(\boldsymbol{W},\boldsymbol{b},\boldsymbol{Z}^{(k-1)},\boldsymbol{U}^{(k-1)}) \tag{10a}$$

$$\boldsymbol{Z}^{(k)} := \underset{\boldsymbol{Z}}{\arg\min}\, \mathcal{L}(\boldsymbol{W}^{(k)},\boldsymbol{b}^{(k)},\boldsymbol{Z},\boldsymbol{U}^{(k-1)}) \tag{10b}$$

$$\boldsymbol{U}^{(k)} := \boldsymbol{U}^{(k-1)} + \boldsymbol{W}^{(k)} - \boldsymbol{Z}^{(k)}. \tag{10c}$$

The problem (10a) is equivalent to:

$$\min_{\boldsymbol{W},\boldsymbol{b}} f(\boldsymbol{W},\boldsymbol{b};\mathcal{D}) + \textstyle\sum_{i=1}^N \frac{\rho_i}{2}\|\boldsymbol{W}_i - \boldsymbol{Z}_i^{(k-1)} + \boldsymbol{U}_i^{(k-1)}\|_F^2. \tag{11}$$

The first term in (11) is a standard CNN loss while the second term is quadratic and differentiable. Thus, this subproblem can be solved by classic stochastic gradient descent.

After solving problem (10a) at iteration k, we proceed to solving problem (10b), which is equivalent to:

$$\min_{\boldsymbol{Z}} \textstyle\sum_{i=1}^N g(\boldsymbol{Z}_i) + \sum_{i=1}^N \frac{\rho_i}{2}\|\boldsymbol{W}_i^{(k)} - \boldsymbol{Z}_i + \boldsymbol{U}_i^{(k-1)}\|_F^2. \tag{12}$$

As $g(\cdot)$ is the indicator function of the constraint set S_i, problem (12) is equivalent to:

$$\boldsymbol{Z}_i^{(k)} = \Pi_{S_i}\left(\boldsymbol{W}_i^{(k)} + \boldsymbol{U}_i^{(k-1)}\right), \tag{13}$$

where Π_{S_i} is the Euclidean projection of $\boldsymbol{W}_i^{(k)} + \boldsymbol{U}_i^{(k-1)}$ onto the set S_i. The special structure of S_i allows us to find the optimal analytical solutions. For filter pruning (Eq. (5)), the solution can be obtained by first calculating

$$O_p = \|[\boldsymbol{W}_i^{(k)} + \boldsymbol{U}_i^{(k-1)}]_{p,:}\|_2^2, \text{ for } p = 1,\cdots,P_i,$$

then keeping α_i rows in $\boldsymbol{W}_i^{(k)} + \boldsymbol{U}_i^{(k-1)}$, corresponding to the α_i largest values in $\{O_p\}_{p=1}^{P_i}$, and setting the rest to zero. For column pruning (Eq. (6)), a similar solution can be obtained by first calculating

$$O_q = \|[\boldsymbol{W}_i^{(k)} + \boldsymbol{U}_i^{(k-1)}]_{:,q}\|_2^2, \quad \text{for } q = 1,\cdots,Q_i,$$

then keeping α_i columns in $\boldsymbol{W}_i^{(k)} + \boldsymbol{U}_i^{(k-1)}$ with the α_i largest values in $\{O_q\}_{q=1}^{Q_i}$, and setting the rest to zero.

C. Final Masked Retraining

The parameters $(\boldsymbol{W},\boldsymbol{b})$ produced by ADMM satisfy the constraints $\{S_i\}_{i=1}^N$ only asymptotically. As a result, retraining process is typically required to improve the accuracy of the pruned model with the training dataset and attain feasibility [40], [42], [45]. To that end, we first construct a binary mask $\boldsymbol{M}_i \in S_i \cap \{0,1\}^{P_i \times Q_i}$ for each layer i to constrain the retraining process. The mask $\boldsymbol{M}_i$ is constructed as follows for filter pruning: first, we compute $\bar{\boldsymbol{Z}}_i = \Pi_{S_i}(\boldsymbol{W}_i), i \in \{1,...,N\}$; then, we set $[\boldsymbol{M}_i]_{p,:} = \mathbf{1}$, for all p s.t. $[\bar{\boldsymbol{Z}}_i]_{p,:\neq 0}$. A similar construction applies for column pruning. We then retrain $\boldsymbol{W}$ using gradient descent but constrained by masks $\{\boldsymbol{M}_i\}_{i=1}^N$. That is, during back propagation, we first calculate the gradient $\nabla_{\boldsymbol{W}_i} f(\boldsymbol{W},\boldsymbol{b};\mathcal{D})$, which is the same as the standard CNN training process, then apply the mask $\boldsymbol{M}_i$ to the gradient using element-wise multiplication. Therefore, the weight update in every step during the retraining process is

$$\boldsymbol{W}_i := \boldsymbol{W}_i - \eta \boldsymbol{M}_i \circ \nabla_{\boldsymbol{W}_i} f(\boldsymbol{W},\boldsymbol{b};\mathcal{D}), \tag{14}$$

where η is the learning rate and $\circ$ denotes element-wise multiplication.

D. Progressive Pruning

When pursuing extremely high pruning rates ($> 90\times$), the weight pruning approach described in Sections IV-B and IV-C has certain drawbacks. First, as feasibility is only asymptotic, many weights are approximately (but not exactly) equal to zero at the conclusion of ADMM. Second, rounding these to zero after the termination of the algorithm often leads to accuracy loss [12], even if one retrains using only the resulting sparse

weights (as described in Section IV-C). To attain high pruning rates with negligible accuracy loss, we follow a progressive weight pruning approach [12] that reaches a high pruning rate gradually.

This progressive pruning is illustrated in Fig. 4. We prune the network over multiple rounds. In each round, we execute the ADMM weight pruning algorithm (Eq. (10)) and a masked retraining process (Eq. (14)) to the pre-trained model. The input to each round r is a pre-trained neural network $\{\boldsymbol{W}_i^{r-1}, \boldsymbol{b}_i^{r-1}\}_{i=1}^N$, obtained as the output from the previous round, as well as a set of constraints $\{S_i^r\}_{i=1}^N$. As the number of rounds increases, stronger constraints are provided and added to the model. Taking filter pruning as an example, recall from Section IV-A that S_i^r is defined by Eq. (5), a stronger constraint S_i^{r+1} should satisfy that $\alpha^r \leq \alpha_i^{r+1}$, for all $i = \{1, \cdots, N\}$. The pre-trained model is thus pruned progressively and reaches the high pruning rate gradually. The output of the round r is a pruned model, consisting of the weight matrix $\mathbf{W}^r$ that satisfies $\mathbf{W}_i^r \in S_i^r, i = 1, ..., N$. We set the input to the first round to be a model pre-trained without any constraints.

An alternative execution of progressive pruning is as follows. Besides the pruned model, each round could output a mask $\{\boldsymbol{M}_i^r\}_{i=1}^N$, as described in Section IV-C, that contain binary-valued elements with value 0 corresponding to the pruned weights and value 1 corresponding to the remaining weights. The framework can pass this mask along with the pruned model to next round. We could then prune the model additionally conditioning on this mask passed. If we do this, the non-zero entries of the pruned weights $\mathbf{W}_i^r$ will be a subset of those of $\mathbf{W}_i^{r-1}$ in the next round. In practice, it is not necessary to pass masks this way. Instead, we may allow the algorithm reversing decisions made in previous rounds and provide it with more freedom on which weights to prune. In our implementation, we perform three rounds of column pruning, without masks passed forward, and one round of filter pruning, with a mask. The latter incorporates filter-wise constraints while optimizing over column-wise constraints. We describe these in detail in Section VI-A.

V. Hardware Implementations

To explore the facilitation of hardware implementation on inference efficiency, we execute the pruned model on three hardware-processing element, Smartphone GPU&CPU, NVIDIA TX2, and FPGA. In this section, we provide a detailed description of these three platforms and our modifications to match the RF fingerprinting task.

A. Smartphone Acceleration Framework

We rely on and extend a compiler-assisted acceleration framework on general, off-the-shelf mobile devices. The compiler-assisted framework is written in C++ and uses the compiler-level concept of *automatic code generation*. It takes the Open Neural Network Exchange (ONNX) model as input, and automatically generates C++ based and OpenCL based executables for mobile CPU and mobile GPU, respectively. Similar to [62], [63], [84], the ONNX model is first translated into computational graphs and weights at compiler level. Then layerwise representation for CNN inference is extracted from the computation graph, supporting layerwise code generation and optimization for the final program executables.

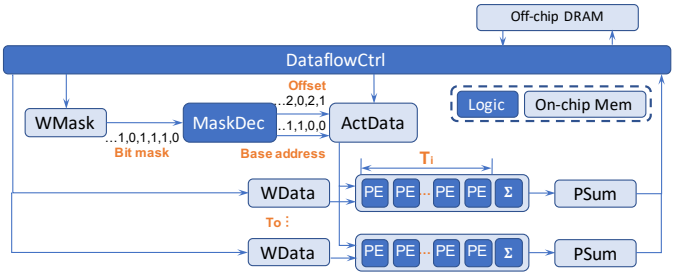


Fig. 5. A dedicated hardware architecture designed for the proposed 1D conv. with the structured pruned model.

We extend the compiler-assisted framework by supporting different types of structured pruning, e.g., filter pruning and column pruning. After such structured pruning, the layer-wise computation is still in the form of GEMM [85], [86]. The computation graph based code generation framework is applied, and parallelism in CNN inference can be maintained without heavy control-flow dependencies. Besides, we adopt two compiler-level optimization techniques that work for both mobile CPU and GPU code generation: *Vectorization* and *Parameter Auto-Tuning*.

Vectorization achieves high parallelism on multi-core mobile CPU/GPU. We use ARM NEON and OpenCL to auto-vectorize CPU and GPU codes, respectively. The mobile CPU and GPU have different numbers of vector registers. In order to take full advantage of registers while avoiding register spilling, we have designed a specific loop unrolling level to pack computations and perform mapping accordingly. Parameter auto-tuning specifically tests candidate configurations of the key performance parameters including: strategies of placing data on CPU/GPU memories, different tile sizes, and loop permutations for each CNN layer on CPU/GPU. Again this is layerwise optimization with the flexibility of different configurations for different layers of the target CNN, and the best layerwise configuration is chosen accordingly.

B. FPGA Framework

Organization of Parallel Processing Elements (PEs). To further improve the computational efficiency when performing inference on an FPGA, we propose a dedicated accelerator for our structured pruned model, leveraging the 1D convolutional structure of our layers. As shown in Fig. 5, the processing elements (PEs) are organized in a 2D array, where each PE executes one multiply-accumulate (MACC) operation. To propose a general architecture design that can be implemented on FPGA devices with different hardware resource, we parameterize the PE array with a size of $T_o \times T_i$; the horizontal T_i PEs compute the same filter, and thus the MACC results are accumulated ($\sum$ in Fig. 5). In contrast, the PE-rows compute T_o filters concurrently. Multiple on-chip memories are connected to the PE array for dataflow.

Dataflow with Dedicated Memory Hierarchy. Due to the limited size of on-chip memory, the `DataflowCtrl` module exchanges the weights and input/output activations between the on-chip memory and off-chip DRAM. The memory structures and the corresponding workflow are organized as follows. In one iteration, the weight values of T_o filters are loaded from DRAM and written to `WData` memory corresponding to each PE-row. The zero values by *column pruning* are emitted in the *compressed* weight stream. The weight mask stored in `WMask` memory indicates the *index* of the skipped column. We use a *bit mask* (1-bit for each column) to represent whether the column is pruned or not. The stream of bit masks is sent to a mask decoder module (`MaskDec`) that translates the codes to *offset* and *base address* in reading the proper activation data in `ActData` memory. In the 1D convolution with a filter size of 3, for instance, the bit mask (binary) stream "0,1,1,1,0,1,..."(as in Fig. 5) can be grouped to 3-bits tuples and each tuple indicates the pruning pattern of an input channel. In this example, tuple "{0,1,1}" for the first channel indicates the first sub-column is pruned and the rest two sub-columns are remained. This stream is then translated to an *offset stream* of "1,2,0,2,...", where "1,2" are the index of non-zero sub-columns corresponding to the first channel and "0,2" are for the second one. Therefore, the *based address stream* "0,0,1,1,..." is output simultaneously and aligned to each element in offset stream to indicate the index of input channel. With the base address and offset, the proper activation values are read from `ActData` and broadcast to T_o PE-rows. Note that T_i bit non-zero bit masks are decoded in each cycle that fetch multiple activation values for T_i PEs concurrently. The `PSum` memory stores the partial sum in computation with multiple input channels. This workflow iterates to get all output filters in one layer and then stores the results back to DRAM for the subsequent layers.

C. TX2 Framework (TensorRT)

We also implement our neural network on an edge GPU NVIDIA Jetson TX2 (see Section VI-A for more details on the platform). We run our methods using two frameworks: Pytorch, which is the standard implementation we also use during training, and TensorRT. TensorRT is briefly described below. We note that both of these frameworks implement pruned networks via masks of zeros and ones; as such, they do not a priori exploit the sparse structure of our pruned network. We thus only use Pytorch and TensorRT at the TX2 for comparison purposes.

TensorRT is a compression and optimization framework developed by NVIDIA for running fast inference on NVIDIA GPUs [68]. After the developer trains and saves the neural network as a Pytorch model, the model can be optimized and run for inference through TensorRT. We use ONNX [68] parser to import the model from its saved format into TensorRT. TensorRT's optimization tool makes several platform-specific optimizations. It merges concatenation layers by directing layer outputs to the correct eventual destination and eliminates the layers whose outputs are not used. It also fuses convolution, bias and ReLU operations together. TensorRT also aggregates operations with sufficiently similar parameters and the same source tensor (for example, the 1x1 convolutions in GoogleNet v5's inception module) and removes the operations which are equivalent to no-ops.

VI. Performance Evaluation

In this section, we discuss results on five benchmark datasets of device transmissions and report the effectiveness of model pruning and the inference speedup on different platforms.

A. Experimental Setup

Datasets. We use five benchmark datasets, summarized in Table I, including three WiFi datasets (WiFi-50, WiFi-Eq-50, WiFi-Eq-500), one ADS-B dataset (ADS-B-50), and one mixture dataset (Mixture-50) containing both WiFi and ADS-B transmissions. WiFi-50 and ADS-B-50 both contain transmissions of raw IQ samples, while WiFi-Eq-50 and WiFi-Eq-500 are partially equalized, as described in Section III-B. We pick 25 devices uniformly at random (u.a.r.) from WiFi-50 and ADS-B-50, respectively, and form Mixture-50. To analyze the effect of model pruning on equalized transmissions, we also perform partial equalization on WiFi transmissions and construct WiFi-Eq-50 based on WiFi-50. To measure the pruning ability with respect to scaling the number of devices the model classifies, we also perform partial equalization on the whole WiFi dataset (500 devices) and form WiFi-Eq-500. Recall from Section III-B that equalization involves down-sampling; as a result, equalized transmissions are shorter than raw transmissions.

Each benchmark dataset has a training set and a test set, containing 218 and 55 transmissions per device, respectively. We further separate 10% of the training set as a validation set to tune hyper-parameters, such as slice and batch sizes. During training and pruning, we evaluate the model on the validation set and keep the best mong all epochs, which is finally evaluated on the test set.

Evaluation Metrics. We evaluate the model performance on five benchmark datasets via *Top-1 Accuracy*, considering that classes are balanced. We report per-slice and per-transmission accuracy, where transmission predictions happen as described in Section III-D.

We evaluate the model pruning performance via *pruning rate*, which is the ratio of unpruned size versus pruned size. Formally, for n and n_0 the total number and zero parameters, respectively, the pruning rate is $\frac{n}{n-n_0}$. We calculate FLOPS (floating point operations per second) for pre-trained and pruned models as well via the THOP Python module. We also evaluate inference acceleration for the pruned models on different platforms (described below in "Software and Hardware"). Though both per-slice and per-transmission inference acceleration are interesting, since the length of transmissions is arbitrary, to make a fair and more general comparison, we report the inference time per slice.

Parameters. We use the following hyper-parameters, which we have determined using the validation set. We use a batch size of 256 and 128 in ResNet50-1D model for WiFi-Eq-50 and other datasets respectively. We use a slice size of 512 for raw datasets (WiFi-50, ADS-B-50, Mixture-50), and

TABLE I
RF FINGERPRINTING DATASET STATISTICS

Datasets	L	N	M	$\bar{W}$	Per-trans. Accuracy			
					Random Guess	RFNet	VGG16-1D	ResNet50-1D
WiFi-50	50	218	55	13071	2%	57.53%	58.13%	**64.80%**
ADS-B-50	50	218	55	9526	2%	**91.85%**	87.08%	88.53%
Mixture-50	50	218	55	11427	2%	74.72%	73.92%	**79.51%**
WiFi-Eq-50	50	218	55	514	2%	63.94%	64.31%	**70.78%**
WiFi-Eq-500	500	218	55	662	0.2%	47.82%	53.77%	**61.40%**

Here, L is the number of devices/labels, N is the number of training transmissions per device, M is the number of test transmissions per device, $\bar{W}$ is the average number of samples per transmission. We report the per-transmission accuracy of non-pruned ResNet50-1D models as well as the random guess, RFNet, VGG16-1D for reference purpose. RFNet [8] is a custom-designed model inspired by AlexNet [23], which contains ten convolution layers and five max pooling layers, organized in five stacks, followed by four fully connected layers. VGG16-1D is a modified version of the VGG16 [32] by setting each filter width to 1×3.

TABLE II
THREE SPARSITY SETTINGS FOR RESNET50-1D

Depth	Sparsity Ratio			Depth	Sparsity Ratio		
	I	II	III		I	II	III
1	0%	0%	0%	27	65%	85%	95%
2-3	30%	50%	60%	28-39	60%	80%	90%
4-5	50%	70%	80%	40-41	62%	82%	92%
6-13	55%	75%	85%	42-43	65%	85%	95%
14-16	60%	80%	90%	44-48	60%	80%	90%
17-20	65%	85%	95%	49	50%	70%	80%
21-26	68%	88%	98%	Prun. Rate	2.6×	5.4×	11.8×

Each column of the table describes the *sparsity ratio* for different layers i, defined as $1-\alpha_i/P_i$ and $1-\alpha_i/Q_i$ for filter and column pruning respectively. The resulting pruning rate $n/(n-n_0)$ for CLs is shown in the last row.

TABLE III
AN ILLUSTRATION OF PROGRESSIVE WEIGHT PRUNING PROCEDURE.

Models	Rounds	Description	Pruning Rate (CLs only)
V1	1	C (I)	2.6×
V2	2	C (I) → C (II)	5.4×
V3	3	C (I) → C (II) → C (III)	11.8×
V4	4	C (I) → C (II) → C (II) + F (II)	27.2×

We present 4 different versions of the execution (V1-V4). In the first three (V1-V3), Column (C) pruning, is sequentially applied with more stringent constraints (I–III) in each round, without masks. In V4, a final Filter (F) pruning round is applied, further constrained via masks.

a smaller slice size 198 for WiFi equalized datasets, due to the shorter equalized transmission length caused by down-sampling. We use Adam [87] as an optimizer with default values and initialize the learning rate to 0.0001. To reduce the evaluation cost, we set the $\kappa = 16$ and a stride of 16 for slicing at training and testing stage, respectively.

ADMM In each model pruning round, we run 50 iterations of ADMM (Eq. (10)). In each iteration, step (10a) is implemented by one epoch of SGD over the dataset, solving Eq. (11) approximately. We set all $\rho_i = 10^{-4}$ initially; every 10 iterations of ADMM, we multiply them by a factor of 10, until they reach 1. Finally, we retrain the network under a pruned mask for 10 epochs with a batch size of 64 as described in Section IV-D.

Progressive Model Pruning. Recall from Section IV-A that the sparsity constraint sets $\{S_i\}_{i=1}^N$ are defined by Eq. (5) for filter pruning and by Eq. (6) for column pruning, with sparsity parameters $\alpha_i \in \mathbb{N}$ determining the non-zero rows or columns, respectively per layer. We define three different sparsity settings (I–III) for ResNet50-1D, summarized in Table II. In each setting we indicate the *sparsity ratio* for different layers i, defined as $1 - \alpha_i/P_i$ and $1 - \alpha_i/Q_i$ for filter and column pruning respectively.

To set the sparsity ratio for each layer, we built upon the intuition/lessons learned from pruning ResNet50 over image classification on the CIFAR10 dataset, reported in [10], [40], [49]. In particular, these works empirically observed that sparsity ratios should should be higher on middle layers, that are less sensitive to pruning, while earlier and later layers should lower pruning rates, as they have greater impact on pruning. To that end, we started from the sparsity ratios reported in [49]; for each group in Table II, we fine-tuned these selections to our RF-fingerprinting dataset. In particular, we explored 5% perturbations of the values reported in [49], and selected the best performing values w.r.t. validation set accuracy. We stress again that the sparsity constaint sets are applied to all CLs, including the CLs in projection and identity blocks, as well as the shortcuts in projection blocks, as shown in Fig 2.

These constraints are used in different rounds of progressive pruning, as illustrated in Fig. 4. We have 4 different versions of this execution (V1-V4) summarized in Table III. In the first three (V1-V3), Column (C) pruning, is sequentially applied with more stringent constraints (I–III) in each round, without masks (see Section IV-D). In V4, a Filter (F) pruning round is applied, further constrained via masks $\{M_i^2\}_{i=1}^N$ from the column layer. In all cases, the first round operates on a pre-trained ResNet50-1D model without constraints. Note that, in V4, the combined application of Column and Filter pruning results in a model with final overall pruning rate 27.2× for CLs.

Software and Hardware. We use GNU Radio Toolkit [88] for data equalization. Training, pruning and prediction are implemented in Python using Pytorch and NVIDIA CUDA support. All training and pruning experiments are carried out on an NVIDIA DGX workstation with 1 Tesla V100 GPU with 32 GB memory and 5120 cores, operating at a frequency of

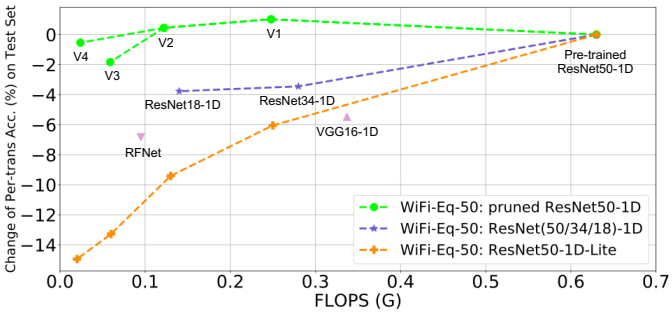
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3064466, IEEE Transactions on Mobile Computing

TABLE IV
PROGRESSIVE MODEL PRUNING ON WIFI-EQ-50 AND WIFI-EQ-500 DATASETS

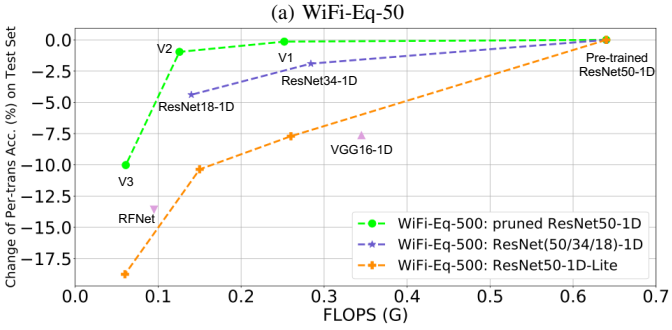
Datasets	Benchmark	Accuracy		Pruning Rate		# Parameters		FLOPS
		Per-slice	Per-trans.	CLs only	All	CLs only	All	
WiFi-Eq-50	ResNet50-1D	85.71%	70.78%	1×	1×	15.90M	16.06M	0.63G
	progressive pruning (V1)	87.47%	71.80%	2.6×	2.5×	6.15M	6.30M	0.24G
	ResNet50-1D-Lite	82.62%	64.73%	-	-	6.22M	6.36M	0.25G
	progressive pruning (V2)	87.44%	71.23%	5.4×	5.1×	2.97M	3.13M	0.12G
	ResNet50-1D-Lite	79.63%	61.37%	-	-	3.05M	3.23M	0.13G
	progressive pruning (V3)	84.77%	68.95%	11.8×	10.5×	1.38M	1.54M	0.06G
	ResNet50-1D-Lite	76.29%	57.50%	-	-	1.39M	1.55M	0.06G
	progressive pruning (V4)	85.59%	70.24%	27.2×	15.8×	0.59M	0.74M	0.02G
	ResNet50-1D-Lite	80.05%	55.83%	-	-	0.59M	0.74M	0.02G
WiFi-Eq-500	ResNet50-1D	45.38%	61.40%	1×	1×	15.90M	16.98M	0.64G
	progressive pruning (V1)	44.83%	61.26%	2.6×	2.1×	6.15M	7.18M	0.25G
	ResNet50-1D-Lite	42.32%	53.69%	-	-	6.21M	7.26M	0.26G
	progressive pruning (V2)	44.51%	60.44%	5.4×	4.3×	2.97M	3.99M	0.13G
	ResNet50-1D-Lite	37.86%	51.04%	-	-	3.33M	4.35M	0.15G
	progressive pruning (V3)	38.77%	51.38%	11.8×	9.2×	1.38M	2.41M	0.06G
	ResNet50-1D-Lite	30.11%	42.64%	-	-	1.30M	2.33M	0.06G

We also evaluate a group of ResNet50-1D-Lite models, which contain fewer filters in each CLs while maintain the ResNet50-1D structure. These ResNet50-Lite models are deliberately designed to have similar number of parameters as pruned models.

1230 MHz and 250W peak power. Besides NIVIDIA V100, we also evaluate the dedicated architectures on a TX2 platform, a smartphone, and an FPGA. NVIDIA Jetson TX2 is an industrial-graded GPU aided computer designed for GPU computations. It features an integrated NVIDIA Pascal GPU with 256 CUDA cores, a hex-core ARMv8 64-bit CPU complex, and 8GB of LPDDR4 memory with a 128-bit interface, at 7.5W peak power and Max-Q frequency of 854 MHz. Our smartphone experiments are conducted on a Samsung Galaxy S10 cell phone with the latest Qualcomm Snapdragon 855 mobile platform that contains a 2.8 GHz Qualcomm Kryo 485 Octacore CPU with 8 cores and a Qualcomm Adreno 640 GPU, operating at a frequency of 585 MHz. The peak power consumption of Qualcomm Snapdragon 855 mobile platform is 5W. We use the FPGA platform Xilinx-ZCU104 [89], with an operating frequency of 200MHz and 14W peak power consumptions, under a 256-PE design ($T_i = 4, T_o = 64$, as described in Section V-B). To explore performance gains in detail, we build a cycle-level emulator for the proposed design and connect it to a DRAM simulation model [90].



(a) WiFi-Eq-50



(b) WiFi-Eq-500

Fig. 6. Performance comparison between the pruned ResNet50-1D, full ResNet(50/34/18)-1D, RFNet [8], VGG16-1D, and ResNet50-1D-Lite on (a) WiFi-Eq-50 and (b) WiFi-Eq-500 datasets. Each point on the plot represents the change of per-transmission accuracy of the corresponding model, when compared to a (non-pruned) pre-trained ResNet50-1D, vs. the pruned model's number of FLOPS. The pruned model outperforms full ResNet(34/18)-1D and its corresponding 'Lite' version in all cases, attaining a considerably higher accuracy at even lower FLOPS.

B. Performance Evaluation

Effectiveness of Model Pruning. We first apply our progressive structured pruning method on the WiFi-Eq-50 and WiFi-Eq-500 datasets. Predictions (per-slice and per-transmission accuracy) and pruning performance (pruning rate, number of model parameters, and FLOPS) are summarized in Table IV. We report the pruning rate and number of model parameters in terms of convolutional layers (CLs only) as well as the entire model (All). We stress again that CLs are of greater impact on inference time than remaining parameters, as they are used repeatedly during inference. Overall, *our approach prunes a large fraction of weights with only minimal test accuracy degradation*. In particular, progressive structured pruning yields a 27.2× pruning rate on CLs, with only 0.54% per-transmission accuracy drop on WiFi-Eq-50 (under V4), and a 5.4× pruning rate on CLs with only 0.96% per-transmission

TABLE V
PROGRESSIVE MODEL PRUNING ON PROTOCOL-MIXTURED DATASETS

Datasets	Benchmark	Accuracy		Pruning Rate		# Parameters		FLOPS
		Per-slice	Per-trans.	CLs only	All	CLs only	All	
WiFi-50	ResNet50-1D	44.52%	64.80%	1×	1×	15.90M	16.36M	1.61G
	progressive pruning (V1)	44.83%	64.82%	2.6×	2.5×	6.15M	6.61M	0.63G
	progressive pruning (V2)	44.74%	64.44%	5.4×	4.8×	2.97M	3.43M	0.31G
	progressive pruning (V3)	44.43%	64.00%	11.8×	8.9×	1.38M	1.85M	0.15G
	progressive pruning (V4)	44.23%	63.61%	27.2×	15.7×	0.59M	1.05M	0.07G
ADS-B-50	ResNet50-1D	73.83%	88.53%	1×	1×	15.90M	16.36M	1.61G
	progressive pruning (V1)	73.55%	88.49%	2.6×	2.5×	6.15M	6.61M	0.63G
	progressive pruning (V2)	73.40%	88.25%	5.4×	4.8×	2.97M	3.43M	0.31G
	progressive pruning (V3)	73.14%	88.18%	11.8×	8.9×	1.38M	1.85M	0.15G
	progressive pruning (V4)	72.86%	88.09%	27.2×	15.7×	0.59M	1.05M	0.07G
Mixture-50	ResNet50-1D	66.19%	79.51%	1×	1×	15.90M	16.36M	1.61G
	progressive pruning (V1)	65.94%	79.55%	2.6×	2.5×	6.15M	6.61M	0.63G
	progressive pruning (V2)	65.86%	79.13%	5.4×	4.8×	2.97M	3.43M	0.31G
	progressive pruning (V3)	65.27%	78.60%	11.8×	8.9×	1.38M	1.85M	0.15G
	progressive pruning (V4)	64.79%	78.01%	27.2×	15.7×	0.59M	1.05M	0.07G

For Mixture-50 dataset, the per-transmission accuracy (79.51%) is slightly above the average of it on WiFi-50 (64.80%) and ADS-B-50 (88.53%); and the per-transmission accuracy after the progressive pruning (78.01%) is still higher than the average of 63.61% and 88.09% on WiFi-50 and ADS-B-50, respectively. These two observations indicate that the informational capacity of the pruned network is not adversely affected by the exposure to protocol mixtures.

TABLE VI
EFFECTIVENESS OF PROGRESSIVE PRUNING

Benchmark	Accuracy		Pruning Rate	
	Per-slice	Per-trans.	CLs only	All
ResNet50-1D	85.71%	70.78%	1×	1×
V4	85.59%	70.22%	27.2×	15.8×
direct pruning	79.32%	61.02%	27.2×	15.8×

Performance comparison between a progressive pruned model (V4) and a single-round pruned model (directly to 27.2×) on WiFi-Eq-50 dataset. Progressive pruning improves accuracy by 9.2%.

accuracy drop) on WiFi-Eq-500 (under V2).

We also demonstrate pruning while training, in a progressive fashion, is imperative for maintaining high accuracy while constructing a parsimonious model. To show this, we construct a class of models that has fewer parameters than the original ResNet50-1D, and explore the resulting accuracy-compression trade-off. We term the first class of models as 'ResNet50-Lite': these models have the same architecture ResNet50 but contain fewer filters in each convolutional layer (resulting in fewer parameters in total). These ResNet50-Lite models are deliberately designed to have similar total number of parameters as pruned models V1-V4. To make a fair comparison, we set the number of filters in each layer corresponding to the sparsity settings shown in Table 2. Specifically, the sparsity ratio for each layer i reported in Table 2 is defined as $1-\alpha_i/P_i$, where α_i is sparsity parameter defined in Eq. (3) and (4). For each layer i, We set the number of filters in each ResNet50-1D-Lite model equal to the corresponding α_i. We train these 'Lite' models until no improvement was observed on the validation set (typically, $\sim$20 epochs). The performance of ResNet50-Lite models on WiFi-Eq-50 and WiFi-Eq-500 datasets is presented in Table IV. The pruned model outperforms its corresponding 'Lite' version in all cases, attaining a considerably higher accuracy at even lower FLOPS.

To further demonstrate this, we plot the change of accuracy (i.e., the accuracy achieved by the corresponding model minus the accuracy of the pre-trained ResNet50-1D model) vs. FLOPS of our pruned models and the ResNet-1D-Lite models described above in Fig. 6. As an additional class of parsimonious competitors, we also include the change of accuracy and FLOPS of the ResNet34-1D and ResNet18-1D architectures [25] (which are shallower than ResNet50, and contain fewer parameters). Our pruned ResNet50-1D outperforms ResNet(34/18)-1D and ResNet50-1D-Lite in all cases, clearly demonstrating the advantage of pruning compared to directly training parsimonious models from scratch.

Saturation and Bias vs. Variance Tradeoff. As seen on Table IV there is a slight accuracy increase during pruning in V1, V2, for WiFi-Eq-50. This is because pruning reduces the complexity of the model, and hence, to some extent, avoids overfitting (reducing variance without affecting model bias). However, increasing the pruning rate beyond a critical point can lead to sharp drop in accuracy; this is expected, as reducing the model capacity significantly hampers its expressiveness and starts to introduce bias in predictions. As indicated in Table IV, this critical point occurs on WiFi-Eq-500 at pruning rate $11.8\times$/V3, observed as a sharp accuracy drop (51.38% at $11.8\times$/V3 vs. 60.44% at $5.4\times$/V2). Not surprisingly, this critical point occurs earlier for WiFi-Eq-500 when compared to WiFi-Eq-50, for which no saturation happens for pruning rates up-to $27.2\times$.

Inference on Protocol Mixtures. Table V summarizes performance of progressive structured pruning for ResNet50-1D on WiFi-50 and ADS-B-50 datasets, as well as on Mixture-50,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2021.3064466, IEEE Transactions on Mobile Computing

TABLE VII
INFERENCE ACCELERATION

Datasets	Benchmark	Overall Comp. Rate	Acc. Degra.	NVIDIA V100 5120-cores 1230MHz 250W	NVIDIA TX2 256-cores 854MHz 7.5W		Phone(GPU) 384×2 ALUs 585MHz Max. Power: 5W	Phone(CPU) 8-cores 2.8GHz	FPGA 256-PE 200MHz 14W
				Pytorch (ms)	Pytorch	TensorRT	(ms)	(ms)	(ms)
WiFi-50	ResNet50-1D	1×	0.00%	9.22±0.19	28.93±2.47	11.81±0.88	37.60	63.20	15.60
	pruned (V4)	15.7×	1.19%	9.18±0.16	29.09±3.14	11.62±0.59	11.50	21.17	1.36
ADS-B-50	ResNet50-1D	1×	0.00%	9.24±0.20	29.10±2.66	11.87±0.82	37.26	63.49	15.60
	pruned (V4)	15.7×	0.44%	9.13±0.21	28.79±2.76	11.66±0.89	11.59	21.83	1.36
Mixture-50	ResNet50-1D	1×	0.00%	9.19±0.24	29.16±2.79	12.00±0.95	37.32	63.44	15.60
	pruned (V4)	15.7×	1.50%	9.16±0.22	29.04±3.09	11.46±0.47	11.15	21.30	1.36
WiFi-Eq-50	ResNet50-1D	1×	0.00%	9.15±0.17	29.40±1.26	9.67±0.99	24.07	40.18	6.04
	pruned (V4)	15.7×	0.54%	9.07±0.12	29.34±1.18	9.37±0.78	7.24	16.21	0.53
WiFi-Eq-500	ResNet50-1D	1×	0.00%	9.20±0.18	29.34±1.13	9.45±0.87	24.13	40.32	6.06
	pruned (V2)	4.3×	0.96%	9.18±0.14	29.15±1.07	9.47±0.77	13.05	24.15	2.13

Inference acceleration for the pruned models on four platforms: NVIDIA V100, NVIDIA TX2, a Samsung Galaxy S10, and a Xilinx-ZCU104 FPGA. Among four platforms, V100 maintains the best performance (∼9ms) on non-pruned model. In contrast, FPGA achieves the best performance under a pruned model and attains a speedup of as much as $11.5\times$ over the non-pruned model, demonstrating the benefit of our design over this accelerator. The low power smartphone has the worst inference performance among all platforms; nevertheless, pruning achieves a $3\times$ speedup.

a dataset comprising both protocols. In general, we observe that the accuracy for ADS-B data is higher than for WiFi data, indicating they are easier to identify; this is consistent with observations made, e.g., in [9]. For Mixture-50 dataset, the per-transmission accuracy (79.51%) is slightly above the average of it on WiFi-50 (64.80%) and ADS-B-50 (88.53%); and the per-transmission accuracy after the progressive pruning (78.01%) is still higher than the average of 63.61% and 88.09% on WiFi-50 and ADS-B-50, respectively. These two observations indicate that the informational capacity of the pruned network is not adversely affected by the exposure to protocol mixtures.

Effectiveness of Progressive Pruning. To evaluate the effectiveness of progressive pruning, we present results obtained from a model pruned to $27.2\times$ *in a single round* in Table VI, as opposed to progressively pruning as in setting V4. Progressive pruning improves 9.2% on per-transmission accuracy on WiFi-50-Eq dataset, which demonstrates the effectiveness of progressive vs. direct pruning.

Influence of Equalization. We observe an interesting phenomenon comparing raw WiFi transmissions (Table V) and their equalized versions (Table IV). Specifically, (non-pruned) ResNet50-1D shows much higher per-transmission accuracy for equalized data (70.78%), compared to the raw WiFi transmissions (64.80%). Moreover, progressive pruning on ResNet50-1D is more stable on equalized data, exhibiting a lower per-transmission accuracy drop (0.54%) at large pruning rate ($27.2\times$), compared to the corresponding accuracy drop (1.19%) observed when training over raw data at the same pruning rate. This indicates that equalization helps both inference and compression. Put differently, by removing the effect of the channel from raw IQ samples, the equalized transmissions carry more concentrated features, which require less capacity and can be captured by pruned models (with fewer parameters) effectively.

Hardware Speedup Comparisons. Table VII illustrates the inference acceleration for the pruned models on our four platforms. We list the first two for reference purposes, as they do not support pruned models: evaluations on both V100 and TX2 involve multiplications via masks containing zeros, and therefore do not yield any performance improvements via pruning.

For all platforms, except the FPGA, we evaluate one slice 10^3 times with a batch size of 1 and report the inference speed means and standard deviation. Our FPGA emulation experiments measure clock cycles/clock frequency, so they are deterministic. The CL pruning rate and per-transmission accuracy degradation are also provided for reference.

As shown in Table VII, V100 maintains the best performance (∼9ms) on non-pruned model; this is not surprising, given the power consumption and specifications of DGX machines. In contrast, the FPGA achieves the best performance under a pruned model, outperforming even V100 in this case. It also attains a speedup of as much as 11.5 times over the non-pruned model over the FPGA, demonstrating the benefit of our design over this accelerator. Not surprisingly, the low power smartphone has the worst inference performance among all platforms; nevertheless, our pruning achieves an almost $3\times$ speedup in this platform as well. We note again that no speedup is observed on V100 and TX2 hardware, as these platforms do not support pruned models.

We also make a comparison over different platforms. Using the numbers reported on Table VII, the dedicated FPGA hardware achieves the best efficiency, as its processing time of the V4-pruned model over WiFi-Eq-50 is 0.53s: this is $17\times$, $18\times$, and $14\times$ better than the V100, TX2-TensorRT and smartphone-GPU, respectively. This is due to both the model compression as well as the fully customized hardware for the pruned model outlined in Section V-B. In particular, the compression storage with bit mask and its decoder logic not

only reduces the bandwidth requirement on off-chip DRAM, but also avoids the branching overhead (for zero-column skipping) present in the GPU/CPU cases.

VII. Conclusions

In this paper, we study radio frequency fingerprinting deployments at resource-constrained edge devices. We use structured pruning to jointly train and sparsify neural networks tailored to edge hardware implementations. Under only negligible accuracy loss (less than 1%), we can achieve at most $27.2\times$ pruning rate on overall convolutional layers for 50-device classification; this is reduced to $5.4\times$ for 500 devices. We demonstrate the efficacy of our approach over multiple edge hardware platforms, including a Samsung Gallaxy S10 phone and a Xilinx-ZCU104 FPGA. Our method yields significant inference speedups, $11.5\times$ on the FPGA and $3\times$ on the smartphone, as well as high efficiency.

Variability in wireless channel conditions and SNR levels have been observed as two major contributors to accuracy degradation in CNN-based RF fingerprinting [8], [27]. Data augmentation tailored to RF signals has been been shown to ameliorate both (see, e.g. [91]). In short, one can expose the CNN to many simulated channel and noise variations that are not present in the original training set. By doing this, the CNN will be more robust and less affected by the presence of unseen channels/noise in the test set. This approach can be combined with pruning to increase robustness against channel variations.

Incorporating additional compression techniques, such as quantization [92], has the potential of improving network efficiency even further; such approaches can be incorporated as additional constraints in our ADMM framework. In addition, they can also yield to hardware-friendly implementations. Investigating such extensions, further driving inference efficiency, is an interesting future direction for this work. Another promising future direction is to explore other network architectures that may be appropriate for fingerprinting tasks on the edge, providing stable and high accuracy with high speed; sequence-based models (e.g., LSTMs) [93], [94] with pruning applied to RF fingerprinting tasks are one such example.

Finally, departing from the standard ML setting, in which the devices in training and test set are the same and a priori known, is an additional interesting future direction. Techniques such as new device detection [77], lifelong learning [81], and open world discovery [95] go beyond the standard setting we consider here; applying pruning to such settings in the context of RF fingerprinting, is an important open question.

References

[1] B. Korany, C. Karanam, H. Cai, and Y. Mostofi, "Xmodal-id: Using wifi for through-wall person identification from candidate video footage," in *Proceedings of the 25th ACM International Conference on Mobile Computing and Networking*, 2019.

[2] D. Zanetti, S. Capkun, and B. Danev, "Types and origins of fingerprints," in *Digital Fingerprinting*, 2016, pp. 5–29.

[3] Y. Huang and H. Zheng, "Radio frequency fingerprinting based on the constellation errors," in *APCC*, 2012, pp. 900–905.

[4] K. Merchant, S. Revay, G. Stantchev, and B. Nousain, "Deep learning for rf device fingerprinting in cognitive communication networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 160–167, 2018.

[5] J. Stankowicz, J. Robinson, J. M. Carmack, and S. Kuzdeba, "Complex neural networks for radio frequency fingerprinting," in *WNYISPW*, 2019, pp. 1–5.

[6] E. Mattei, C. Dalton, A. Draganov, B. Marin, M. Tinston, G. Harrison, B. Smarrelli, and M. Harlacher, "Feature learning for enhanced security in the internet of things," in *GlobalSIP*, 2019, pp. 1–5.

[7] G. Baldini, C. Gentile, R. Giuliani, and G. Steri, "Comparison of techniques for radiometric identification based on deep convolutional neural networks," *Electronics Letters*, vol. 55, no. 2, pp. 90–92, 2018.

[8] T. Jian, B. C. Rendon, E. Ojuba, N. Soltani, Z. Wang, K. Sankhe, A. Gritsenko, J. Dy, K. Chowdhury, and S. Ioannidis, "Deep learning for rf fingerprinting: A massive experimental study," in *IEEE Internet of Things Magazine*, 2020.

[9] A. Al-Shawabka, F. Restuccia, S. D'Oro, T. Jian, B. C. Rendon, N. Soltani, J. Dy, S. Ioannidis, K. Chowdhury, and T. Melodia, "Exposing the fingerprint: Dissecting the impact of the wireless channel on radio fingerprinting," in *INFOCOM*, 2020.

[10] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *ECCV*, 2018, pp. 184–199.

[11] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *ASPLOS*, 2019, pp. 925–938.

[12] S. Ye, T. Zhang, K. Zhang, J. Li, K. Xu, Y. Yang, F. Yu, J. Tang, M. Fardad, S. Liu, X. Chen, X. Lin, and Y. Wang, "Progressive weight pruning of deep neural networks using admm," *CoRR*, vol. abs/1810.07378, 2018.

[13] O. Ureten and N. Serinken, "Wireless security through RF fingerprinting," *Canadian Journal of Electrical and Computer Engineering*, vol. 32, no. 1, pp. 27–33, 2007.

[14] W. C. Suski II, M. A. Temple, M. J. Mendenhall, and R. F. Mills, "Radio frequency fingerprinting commercial communication devices to enhance electronic security," *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 3, pp. 301–322, 2008.

[15] V. Lakafosis, A. Traille, H. Lee, E. Gebara, and M. M. Tentzeris, "RF fingerprinting physical objects for anticounterfeiting applications," *IEEE Transactions on Microwave Theory and Techniques*, vol. 59, no. 2, pp. 504–514, 2011.

[16] S. U. Rehman, K. W. Sowerby, and C. Coghill, "Analysis of impersonation attacks on systems using rf fingerprinting and low-end receivers," *Journal of Computer and System Sciences*, vol. 80, no. 3, pp. 591 – 601, 2014.

[17] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, 2008, pp. 116–127.

[18] T. D. Vo-Huu, T. D. Vo-Huu, and G. Noubir, "Fingerprinting Wi-Fi devices using software defined radios," in *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2016, pp. 3–14.

[19] K. Merchant, S. Revay, G. Stantchev, and B. Nousain, "Deep learning for RF device fingerprinting in cognitive communication networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 160–167, 2018.

[20] S. Riyaz, K. Sankhe, S. Ioannidis, and K. Chowdhury, "Deep learning convolutional neural networks for radio identification," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 146–152, 2018.

[21] K. Sankhe, M. Belgiovine, F. Zhou, S. Riyaz, S. Ioannidis, and K. Chowdhury, "ORACLE: Optimized Radio clAssification through Convolutional neuraL nEtworks," in *IEEE International Conference on Computer Communications*, 2019.

[22] T. Jian, B. C. Rendon, A. Gritsenko, J. Dy, K. Chowdhury, and S. Ioannidis, "MAC ID spoofing-resistant radio fingerprinting," in *GlobalSIP*, 2019.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

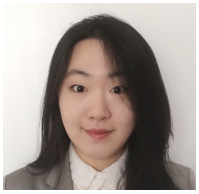
[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[26] F. Restuccia, S. D'Oro, A. Al-Shawabka, M. Belgiovine, L. Angioloni, S. Ioannidis, K. Chowdhury, and T. Melodia, "Deepradioid: Real-time channel-resistent optimization of deep learning-based radio fingerprint-

ing algorithm," in *ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM MobiHoc)*, Catania, Italy, 2019.
[27] H. Jafari, O. Omotere, D. Adesina, H. Wu, and L. Qian, "Iot devices fingerprinting using deep learning," in *MILCOM*, 2018, pp. 1–9.
[28] J. Yu, A. Hu, G. Li, and L. Peng, "A robust rf fingerprinting approach using multisampling convolutional neural network," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6786–6799, 2019.
[29] A. M. Ali, E. Uzundurukan, and A. Kara, "Assessment of features and classifiers for bluetooth rf fingerprinting," *IEEE Access*, vol. 7, pp. 50 524–50 535, 2019.
[30] J. Han, C. Qian, P. Yang, D. Ma, Z. Jiang, W. Xi, and J. Zhao, "Geneprint: Generic and accurate physical-layer identification for uhf rfid tags," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 846–858, 2016.
[31] B. W. Ramsey, T. D. Stubbs, B. E. Mullins, M. A. Temple, and M. A. Buckner, "Wireless infrastructure protection using low-cost radio frequency fingerprinting receivers," *International Journal of Critical Infrastructure Protection*, vol. 8, pp. 27–39, 2015.
[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.
[33] N. Soltani, K. Sankhe, S. Ioannidis, D. Jaisinghani, and K. Chowdhury, "Spectrum awareness at the edge: Modulation classification using smartphones," in *DySPAN*, 2019, pp. 1–10.
[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012, pp. 1097–1105.
[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[36] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.
[37] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *NeurIPS*, 2017, pp. 4857–4867.
[38] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.
[39] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017, pp. 1389–1397.
[40] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.
[41] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV*, 2017, pp. 2736–2744.
[42] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *NeurIPS*, 2016, pp. 2074–2082.
[43] M. Yang, M. Faraj, A. Hussein, and V. Gaudet, "Efficient hardware realization of convolutional neural networks using intra-kernel regular pruning," in *ISMVL*, 2018, pp. 180–185.
[44] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *CVPR*, 2019, pp. 2780–2789.
[45] X. Zhu, W. Zhou, and H. Li, "Improving deep neural network sparsity through decorrelation regularization," in *IJCAI*, 2018, pp. 3264–3270.
[46] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *NeurIPS*, 2018, pp. 875–886.
[47] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *ICCV*, 2017, pp. 5058–5066.
[48] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "Autoslim: An automatic dnn structured pruning framework for ultra-high compression rates," *arXiv preprint arXiv:1907.03141*, 2019.
[49] T. Zhang, K. Zhang, S. Ye, J. Li, J. Tang, W. Wen, X. Lin, M. Fardad, and Y. Wang, "Adam-admm: A unified, systematic framework of structured weight pruning for dnns," *arXiv:1807.11091*, 2018.
[50] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, "Coordinating filters for faster deep neural networks," in *ICCV*, 2017, pp. 658–666.
[51] A. Tulloch and Y. Jia, "High performance ultra-low-precision convolutions on mobile devices," in *NeurIPS*, 2017.
[52] S. Dieleman, J. De Fauw, and et.al., "Exploiting cyclic symmetry in convolutional neural networks," in *ICML*, vol. 48, 2016, pp. 1889–1898.
[53] S. Zhai, Y. Cheng, and et.al., "Doubly convolutional neural networks," in *NeurIPS*, 2016, pp. 1082–1090.
[54] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.
[55] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *Advances in neural information processing systems*, 2017, pp. 742–751.
[56] G. K. Nayak, K. R. Mopuri, and et.al., "Zero-shot knowledge distillation in deep networks," in *ICML*, 2019, pp. 4743–4751.
[57] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*, 2015, pp. 161–170.
[58] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks," *Neural Computing and Applications*, pp. 1–31, 2018.
[59] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *MobiSys*, 2016, pp. 123–136.
[60] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *MobiSys*, 2017, pp. 82–95.
[61] https://www.tensorflow.org/mobile/tflite/.
[62] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *the USENIX Symposium on Operating Systems Design and Implementation*, 2018.
[63] https://github.com/alibaba/MNN/.
[64] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *CVPR*, 2015, pp. 806–814.
[65] P. Hill, A. Jain, M. Hill, B. Zamirai, C.-H. Hsu, M. A. Laurenzano, S. Mahlke, L. Tang, and J. Mars, "Deftnn: Addressing bottlenecks for dnn execution on GPUs via synapse vector elimination and near-compute data fission," in *MICRO*, 2017, pp. 786–799.
[66] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 27–40, 2017.
[67] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *MobiSys*, 2018, pp. 389–400.
[68] Nvidia, "TensorRT developer guide," http://https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html, accessed: 03-15-2020.
[69] X. Li, Y. Zhou, Z. Pan, and J. Feng, "Partial order pruning: for best speed/accuracy trade-off in neural architecture search," in *CVPR*, 2019, pp. 9145–9153.
[70] M. Vandersteegen, K. Van Beeck, and T. Goedemé, "Super accurate low latency object detection on a surveillance uav," in *16th International Conference on Machine Vision Applications*, 2019, pp. 1–6.
[71] H.-H. Wu, Z. Zhou, M. Feng, Y. Yan, H. Xu, and L. Qian, "Real-time single object detection on the uav," in *ICUAS*, 2019, pp. 1013–1022.
[72] E. Sourour, H. El-Ghoroury, and D. McNeill, "Frequency offset estimation and correction in the ieee 802.11a wlan," in *IEEE 60th Vehicular Technology Conference*, vol. 7, 2004, pp. 4923–4927.
[73] https://github.com/bastibl/gr-ieee802-11.
[74] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.
[75] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.
[76] Y. Xian, B. Schiele, and Z. Akata, "Zero-shot learning-the good, the bad and the ugly," in *CVPR*, 2017, pp. 4582–4591.
[77] A. Gritsenko, Z. Wang, T. Jian, J. Dy, K. Chowdhury, and S. Ioannidis, "Finding a 'new'needle in the haystack: Unseen radio detection in large populations using deep learning," in *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2019, pp. 1–10.
[78] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *ICML*, 2017, pp. 3987–3995.
[79] G. M. van de Ven and A. S. Tolias, "Generative replay with feedback connections as a general strategy for continual learning," in *COSYNE Workshop*, 2019.
[80] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," in *ICLR*, 2018.
[81] Z. Wang, T. Jian, K. Chowdhury, Y. Wang, J. Dy, and S. Ioannidis, "Learn-prune-share for lifelong learning," in *20th IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1–10.
[82] 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.
[83] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[84] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *ASPLOS*, 2020.
[85] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *Proceedings of the 19th annual ACM symposium on Theory of Computing*, 1987, pp. 1–6.
[86] K. Goto and R. A. v. d. Geijn, "Anatomy of high-performance matrix multiplication," *ACM Transactions on Mathematical Software*, vol. 34, no. 3, pp. 1–25, 2008.
[87] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: http://arxiv.org/abs/1412.6980
[88] M. Müller. (2018, Aug.) Gnu radio v3.7.13.4 (press release). [Online]. Available: https://www.gnuradio.org/news/2018-07-15-gnu-radio-v3-7-13-4-release/
[89] Xilinx. (2020, mar) Zynq ultrascale+ mpsoc zcu104 evaluation kit. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu104.html
[90] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "Dramsim: a memory system simulator," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 100–107, 2005.
[91] N. Soltani, K. Sankhe, J. Dy, S. Ioannidis, and K. Chowdhury, "More is better: Data augmentation for channel-resilient rf fingerprinting," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 66–72, 2020.
[92] S. Ye, T. Zhang, K. Zhang, J. Li, J. Xie, Y. Liang, S. Liu, X. Lin, and Y. Wang, "A unified framework of DNN weight pruning and weight clustering/quantization using ADMM," *CoRR*, 2018.
[93] S. Wang, P. Lin, R. Hu, H. Wang, J. He, Q. Huang, and S. Chang, "Acceleration of lstm with structured pruning method on fpga," *IEEE Access*, vol. 7, pp. 62 930–62 937, 2019.
[94] X. Dai, H. Yin, and N. K. Jha, "Grow and prune compact, fast, and accurate lstms," *IEEE Transactions on Computers*, vol. 69, no. 03, pp. 441–452, 2020.
[95] Z. Wang, B. Salehi, A. Gritsenko, K. Chowdhury, S. Ioannidis, and J. Dy, "Open-world class discovery with kernel networks," in *20th IEEE International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1–10.



Tong Jian is currently pursuing the Ph.D. degree in the Department of Electrical and Computer Engineering, Northeastern University. She received her M.Sc. (2016) in Electrical Engineering from Rensselaer Polytechnic Institute, NY. She works under the guidance of Prof. Stratis Ioannidis in the field of machine learning. Her current research efforts are focused on the application of machine learning in the domain of wireless communication.

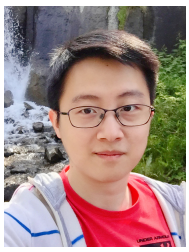
Yifan Gong is currently pursuing the Ph.D. degree under the guidance of Prof. Yanzhi Wang in the Department of Electrical and Computer Engineering, Northeastern University. She received her M.A.Sc degree from Department of Electrical and Computer Engineering, University of Toronto, in 2019. Her current research area focuses on real-time and energy-efficient deep learning and artificial intelligence systems and model compression of deep neural networks.

Zheng Zhan is currently a Ph.D. candidate in the department of Electrical and Computer Engineering of Northeastern University, Boston. He works under the supervision of Prof.Yanzhi Wang.

His research interests include model compression, representation learning, resource management, and Single Image Super-Resolution (SISR).

Runbin Shi received the BEng and MEng degrees from Soochow University, Suzhou, China, in 2013 and 2016, respectively. He has been pursuing his PhD degree with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, since September 2016. His research interest is on modeling and optimization for FPGA-accelerator design.

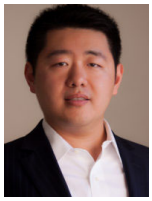
Nasim Soltani is currently a PhD student at the department of Electrical and Computer Engineering of Northeastern University, Boston. She is pursuing her PhD under guidance of Professor Chowdhury in wireless communication. Her current research area focuses on deep learning algorithms for signal classification. She is interested in algorithms and methods for implementing deep learning on resource constrained devices.



Zifeng Wang is currently pursuing the Ph.D. degree in the Department of Electrical and Computer Engineering, Northeastern University, Boston. He received his B.Sc. (2014) in Electronic Engineering from Tsinghua University, China. He works under the guidance of Prof. Jennifer Dy in machine learning. His current research focuses on lifelong learning, representation learning and the application of machine learning in the domain of biostatistics and wireless communication.

Jennifer Dy is Professor at the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, where she first joined the faculty in 2002. She received her M.S. and Ph.D. in 1997 and 2001 respectively from Purdue University, and her B.S. degree from the University of the Philippines in 1993. Her research spans both fundamental research in machine learning and their application to biomedical imaging, health, science and engineering, with research contributions in unsupervised learning, dimensionality reduction, feature selection, learning from uncertain experts, active learning, Bayesian models, and deep representations. She received an NSF Career award in 2004. She has served or is serving as Secretary for the International Machine Learning Society, associate editor/editorial board member for the Journal of Machine Learning Research, Machine Learning journal, IEEE Transactions on Pattern Analysis and Machine Intelligence, organizing and or technical program committee member for premier conferences in machine learning and data mining (ICML, NeurIPS, ACM SIGKDD, AAAI, IJCAI, UAI, AISTATS, SIAM SDM), and program co-chair for SIAM SDM 2013 and ICML 2018.

Kaushik Chowdhury is Associate Professor at Northeastern University, Boston, MA. He was awarded the Presidential Early Career Award for Scientists and Engineers (PECASE), DARPA Young Faculty Award, the Office of Naval Research Early Career Award in 2016, and the NSF CAREER in 2015. He received best paper awards at IEEE GLOBECOM'19, IEEE DySPAN'19, IEEE INFOCOM'18, ACM SenSys'18 (runners up), IEEE ICC'09, '12 and '13, and ICNC'13. He is presently a co-director of the Platforms for Advanced Wireless Research (PAWR) project office. His current research interests involve systems aspects of networked robotics, machine learning for agile spectrum sensing/access, wireless energy transfer, and large-scale experimental deployment of emerging wireless technologies.



Yanzhi Wang is currently an assistant professor at Dept. of ECE at Northeastern University, Boston, MA. He received the B.S. degree from Tsinghua University in 2009, and Ph.D. degree from University of Southern California in 2014. His research interests focus on model compression and platform-specific acceleration of deep learning applications. His recent research achievement, CoCoPIE, can achieve real-time performance on almost all deep learning applications using off-the-shelf mobile devices, outperforming competing frameworks by up to 180X acceleration. His work has been published broadly in top conference and journal venues (e.g., DAC, ICCAD, ASPLOS, ISCA, MICRO, HPCA, PLDI, ICS, PACT, ISSCC, AAAI, ICML, CVPR, ICLR, IJCAI, ECCV, ICDM, ACM MM, FPGA, LCTES, CCS, VLDB, PACT, ICDCS, Infocom, C-ACM, JSSC, TComputer, TCAS-I, TCAD, TCAS-I, JSAC, TNNLS, etc.), and has been cited over 7,400 times. He has received four Best Paper Awards, has another ten Best Paper Nominations and four Popular Paper Awards. He has received ARO Young Investigator Program Award (YIP), Massachusetts Acorn Innovation Award, and other research awards from Google, MathWorks, etc. Three of his Ph.D./postdoc alumni become tenure track faculty at Univ. of Connecticut, Clemson University, and Texas A&M University, Corpse Christi.

Stratis Ioannidis is Associate Professor in the Electrical and Computer Engineering Department of Northeastern University, in Boston, MA, where he also holds a courtesy appointment with the Khoury College of Computer Sciences. He received his B.Sc. (2002) in Electrical and Computer Engineering from the National Technical University of Athens, Greece, and his M.Sc. (2004) and Ph.D. (2009) in Computer Science from the University of Toronto, Canada. Prior to joining Northeastern, he was a research scientist at the Technicolor research centers in Paris, France, and Palo Alto, CA, as well as at Yahoo Labs in Sunnyvale, CA. He is the recipient of an NSF CAREER Award, a Google Faculty Research Award, a Facebook Research Award, a Martin W. Essigmann Outstanding Teaching Award, and best paper awards at ACM ICN 2017 and IEEE DySPAN 2019. His research interests span machine learning, distributed systems, networking, optimization, and privacy.