RBNN: Memory-Efficient Reconfigurable Deep Binary Neural Network With IP Protection for Internet of Things

Huming Qiu, Hua Ma, Zhi Zhang\textsuperscript{\textsuperscript{\textcopyright}}, Yansong Gao\textsuperscript{\textsuperscript{\textcopyright}}, Member, IEEE, Yifeng Zheng\textsuperscript{\textsuperscript{\textcopyright}}, Anmin Fu\textsuperscript{\textsuperscript{\textcopyright}}, Pan Zhou\textsuperscript{\textsuperscript{\textcopyright}}, Senior Member, IEEE, Derek Abbott\textsuperscript{\textsuperscript{\textcopyright}}, Fellow IEEE, and Said F. Al-Sarawi\textsuperscript{\textsuperscript{\textcopyright}}, Senior Member, IEEE

Abstract—Currently, a high demand for on-device deep neural network (DNN) model deployment is limited by the large model size, computing-intensive floating-point operations (FLOPS), and intellectual property (IP) infringements (i.e., easy access to model duplication for the avoidance of license payments). One appealing solution to addressing the first two concerns is model quantization, which reduces the model size and uses integer operations commonly supported by microcontrollers (MCUs usually do not support FLOPS). To this end, a 1-bit quantized DNN model or deep binary neural network (BNN) significantly improves the memory efficiency, where each parameter in a BNN model has only 1 bit. However, BNN cannot directly provide IP protection (in particular, the functionality of the model is locked unless there is a license payment). In this article, we propose a reconfigurable BNN (RBNN) to further amplify the memory efficiency for resource-constrained Internet of Things (IoT) devices while naturally protecting the model IP. Generally, RBNN can be reconfigured on demand to achieve any one of $M$ tasks with the same parameter set, thus only a single task determines the memory requirements. In other words, the memory utilization is improved by a factor of $M$. This work was supported in part by the National Natural Science Foundation of China under Grant 62020461 and Grant 62072239, and Grade 619274248; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20200461 and Grant BK20211192; and in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2021A151510027. This article was recommended by Associate Editor P. A. Beerel. (Huming Qiu and Hua Ma contributed equally to this work.) (Corresponding author: Yansong Gao.)

Huming Qiu, Yansong Gao, and Anmin Fu are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: 120106222682@njjust.edu.cn; yansong.gao@njjust.edu.cn; fuami@njjust.edu.cn).

Hua Ma is with the School of Electrical and Electronic Engineering, The University of Adelaide, Adelaide, SA 5005, Australia, and also with Data61, CSIRO, Sydney, NSW 2122, Australia (e-mail: hua.ma@adelaide.edu.au).

Zhi Zhang is with Data61, CSIRO, Sydney, NSW 2122, Australia (e-mail: zhi.zhang@data61.csiro.au).

Yifeng Zheng is with the School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen 518055, Guangdong, China (e-mail: yifeng.zheng@hit.edu.cn).

Pan Zhou is with the Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China (e-mail: panzhou@hust.edu.cn).

Derek Abbott and Said F. Al-Sarawi are with the School of Electrical and Electronic Engineering, The University of Adelaide, Adelaide, SA 5005, Australia (e-mail: derek.abbott@adelaide.edu.au; said.alsarawi@adelaide.edu.au).

Digital Object Identifier 10.1109/TCAD.2022.3197499


differential privacy (IP) infringements (e.g., SRAM memory power-up pattern). By doing so, an RBNN model can only be used per paid user per authorized device, thus benefitting both the user and the model provider. The source code is released at https://github.com/LearningMaker/RBNN.

Index Terms—Deep neural network (DNN), intellectual property (IP) protection, Internet of Things (IoT), quantization, reconfigurable binary neural network (RBNN).

I. INTRODUCTION

D
eep neural networks (DNNs) today have been deployed in numerous tasks, such as computer vision, natural language processing, and speech recognition. With the trend of machine learning as a service (MLaaS), trained models are put onto the cloud to provide inference services. However, many critical tasks (e.g., autonomous driving) require zero-or minimum-latency in the prediction or inference stage, thus the latency caused by the interaction between the cloud and user is a hurdle. In addition, when a model is hosted on the cloud, it is limited to a fixed level of network bandwidth that might not always be achievable. Further, from the privacy perspective, hosting the model on the cloud poses a threat to data privacy, e.g., data must not leave the device where the user does not want to send the private data to the server. To achieve low-latency, reliable network connectivity, and privacy, a straightforward solution is to host the model on localized pervasive mobile computing platforms such as Internet of Things (IoT) devices.

However, a DNN model has a large number of parameters that are typically stored with a precision of float32, demanding a large model memory footprint. For instance, AlexNet [3] has more than one hundred million parameters that can potentially

Please note that privacy leakage can occur in various vectors in deep learning. Performing model inference on local devices can address the privacy concern where user data are sent to the cloud for inference, while other privacy concerns [1] may still exist, e.g., knowledge distillation-based privacy leakage [2].
result in a model size of 240 MB [4]. Such memory requirements can hardly be satisfied by memory-constrained devices. We note that FPGA is commonly utilized today for accelerating the DNN inference while its on-chip memory can be as low as 10 MB [4]. Furthermore, floating-point operations (FLOPS) are numerous and computing-intensive. Take for example the ResNet-50 [5], which needs about 4.1B FLOPS to process an image of size 224×224×3 [6]. The FLOPS are available in high-performance hardware such as GPUs but may not be (well) supported by low-power IoT devices, such as drones and watches.

The above requirements of large model size and extensive FLOPS lead to heavyweight DNN models as well as notable model inference/prediction latency, thus hindering the DNN model’s ubiquitous deployment on IoT devices. As such, the model must be as lightweight as possible [7] when it is implemented in resource-constrained computing platforms. A practical solution is to reduce model precision by performing so-called model quantization. This approach has attracted significant attention from both academia [8] and industry [9].

A. Limitations

Model Memory Efficiency: Model quantization can be classified into two categories based on the phase in which quantization is introduced: 1) post-training quantization and 2) quantization-aware training [9]. The former quantizes the weight and/or activation after the model has been trained and results in a slight degradation in the inference accuracy, which is usually applicable when quantizing heavyweight DNN models. If a model is designed to be lightweight to cater for IoT devices, then quantization-aware training improves accuracy, compared to post-training quantization. The quantization of these models can range from 8 bit, 4 bit, and even down to 1 bit [10], [11], [12], [13], [14], [15] for not only weight but also activation [16]. Particularly, 1-bit quantized models are referred to as binary neural networks (BNNs) where both the weights and activations are represented by two possible values, -1/0 and +1, substantially reducing the memory footprints. In addition, the FLOPS are replaced by simpler operations, such as the XNOR logical operation and bitcount to expedite the inference [17].

To this end, it seems that the memory efficiency for a given model is maximized by the binarized model. We are now interested in the following research question:

Is it possible to further increase the memory utilization of a given BNN? If so, what evidence can be used to pragmatically demonstrate that?

Model IP Protection: At the same time, on-device model (in particular, BNN in this work) deployment adversely raises a critical security concern. In particular, the unauthorized user may replicate the same model—due to easy access to the model—to other devices or even share the model with many others without proprietary payments, imposing monetary loss to the model provider. This is because the model training could involve incredible computational resources, large proprietary data, and domain expertise, thus possessing significant commercial value to the model provider [18], [19], [20], [21]. Therefore, we are further interested in the following research question:

Is it possible to protect the on-device model intellectual property (IP) by preventing unauthorized usage, while increasing the model memory utilization?

B. Our Solution

We propose a reconfigurable deep BNN (RBNN) to integrate various tasks into the same model, thus further improving the memory utilization to facilitate on-device model deployment, and protecting the model IP at the same time. Our main insight is that multiple tasks can be concurrently learned during the training stage through proper objective functions. Thus, multiple tasks can be trained at the same time on the same model [22]. However, unlike common multitask learning that shares representations among related tasks [22], the RBNN does not share representations and thus can be trained for completely different tasks. By using different model parameter permutations, the RBNN can change the inference task on-demand to serve multiple purposes. Take the smart home as an example, we can train a face recognition task and a speech recognition task in the same RBNN model and both tasks share the same model parameters including weights and activations. At the inference stage, the trained model first serves face recognition—multiple trials can be allowed if inference accuracy is low—to authorize the user to enter the home and then switch to the speech recognition task to serve as a voice assistant. At the same time, we recognize that the reconfigurable nature of the RBNN is a convenient feature for preventing the model being used by unauthorized users or/and devices, as the reconfiguration can be bound up with an authorized key from the model provider to enable IP protection on a deployed RBNN model, that is, only a unique key can properly configure/unlock the model to a specific paid task.

C. Contributions

Our experiments on RBNN are to answer the following research questions.

Q1: Can multiple distinctive tasks (e.g., with a varying number of classes/labels or/and from different domains) be efficiently co-learned during the training, thus enabling RBNN?

Q2: Can the IP protection functionality be enabled or integrated with RBNN?

Building upon the experiments designed for the above questions, the contributions of this work are threefold.

1) We propose RBNN to address memory constraints when deploying a DNN model onto IoT devices to achieve low-latency and low-network connectivity. Instead of training a BNN model for a single task, a single RBNN model can be reconfigured on-demand to serve multiple inference tasks.

2) We comprehensively evaluate RBNN on three common model architectures, including VGG, ResNet, and ReActNet (with ResNet and MobileNet as backbone, respectively) on seven commonly used datasets. We demonstrate that a) tasks with a varying number of
classes can be integrated into the RBNN and b) tasks from two different domains: vision and audio, can co-exist in the RBNN as the model architecture can handle cross-domain tasks. In addition, RBNN is independent of model architectures as validated via three varying model architectures. Our experiments show that RBNN achieves superior memory efficiency and the accuracy for each inference task in the RBNN is on par with its corresponding BNN (Q1).

3) We propose an IP protection framework to bind the RBNN reconfiguration with both a user key and a device key derived from a physical unclonable function (PUF) to prevent executing the model on an unauthorized device, and enabling the IP license to be issued per device and per user (Q2).

The remainder of this article is organized as follows. Section II gives the necessary background related to the presented work. Section III details the realization of the proposed RBNN framework. Section IV comprehensively evaluates the RBNN with various model architectures and datasets. Section V exploits the reconfiguration to naturally protect the RBNN IP by binding the reconfiguration of each task by a unique key. Section VI compares the RBNN with other related designs from the multitask integration and security aspects, respectively, and provides further discussions. This work is concluded in Section VII.

II. BACKGROUND AND RELATED WORK

A. Deep Neural Network

When provided with an input \( x \in \mathbb{R}^n \) with \( n \) dimensions, a DNN \( F_\theta \) will map \( x \) to one of \( C \) classes in common classification tasks. More precisely, the output \( y \in \mathbb{R}^m \) of the DNN is a so-called softmax—a probability vector for \( C \) classes. In other words, \( y_i \) is the probability that the input \( x \) belongs to the \( i \)th class. An input \( x \) is inferred as class \( i \) if \( y_i \) has the highest probability, and thus the output class label \( z \) is \( \text{argmax}_{i \in [1, C]} y_i \) [23].

Generally, the DNN model \( F_\theta \) is trained with a number of samples with annotated ground-truth labels—this is the training dataset. The trainable parameter \( \theta \) comprises weights and biases, which will be learned during the training phase. Specifically, suppose that the training dataset is a set, \( D_{\text{train}} = \{ (x_i, y_i) \}_{i=1}^{S} \) of \( S \) samples, where \( x_i \in \mathbb{R}^n \) and corresponding ground-truth labels \( z_i \in [1, C] \). The training phase permits the learning of parameters that minimize the errors between inputs’ predictions and their ground-truth labels, which can be evaluated by a loss function \( \mathcal{L} \). Parameters \( \Theta \) after training are expressed as

\[
\Theta = \arg \min_{\theta} \sum_{i=1}^{S} \mathcal{L}(F_\theta(x_i), z_i). \tag{1}
\]

Notably, to have accurate inference, the DNN always uses many layers. Thus, a very large number of parameters need to be trained and stored. In the inference stage, the computation and storage overhead are highly related to the size of the DNN model. In addition, the parameters are expressed with float32 precision by default, so the inference is performed based on the intensive FLOPS.

B. Binary Neural Network

In the literature, a number of model compression techniques have been proposed to reduce the memory footprint and the computational overhead. These techniques include pruning or trimming off less important parameters [24], [25], knowledge distillation for transferring knowledge from a large model to a small one [26], and model quantization [27]. This work focuses on the last item. Compared with a full-precision model that uses floating points, a quantized model compresses the original model weights by representing the model weights with lower precision while retaining the inference accuracy. The quantized model can utilize homogeneous-precision or mixed-precision quantization [28]. The ultimate quantization is binarization, resulting in the BNN model, which plays a significant role in maximizing memory utilization for resource-constrained IoT devices.

Binarization is a 1-bit quantization, where a parameter has only two possible values, namely, \(-1/0\) or \(+1\). When training a BNN model, only 1 bit is used to represent both weight \( w \) and activation \( a \) parameters. Following [16], binary functions are expressed as follows:

\[
Q_w(w) = \alpha b_w, \quad Q_a(a) = \beta b_a \tag{2}
\]

where \( b_w \) and \( b_a \) represent the binary weight and activation, and each is multiplied by a scale factor of \( \alpha \) or \( \beta \). The sign functions of \( Q_w \) and \( Q_a \) for binarization are as follows:

\[
\text{sign}(x) = \begin{cases} +1 & x \geq 0 \\ -1 & \text{otherwise.} \end{cases}
\]

After binarization, the floating-point convolution operation in forward propagation can be expressed as

\[
z = \sigma(Q_w(w) \odot Q_a(a)) = \sigma(\alpha \beta (b_w \odot b_a))
\]

where \( \odot \) represents the vector inner product realized by bitwise XNOR and Bitcount operations, which can be performed more efficiently. Thus, binarization not only saves memory footprint but also makes computation much more efficient than floating point operations are replaced by bitwise operations.

C. Model IP Protection

The IP protection of a DNN model is demanded in practice [18], [19], [20], [21]. This is mainly because DNN models are usually trained with computing-intensive resources to process a large amount of proprietary data. In addition, the model training also needs domain expertise. The trained model is valuable for a model provider as an IP and should be protected to preserve its competitive commercial advantage [19].

Watermarking is an approach to embedding provider-known-secret information into the released model to prevent model piracy [18], [19], since later the model provider can use the watermark to claim the ownership. However, watermarking cannot prevent covertly leaked model usage, especially on the localized IoT devices. This trivially bypasses the ownership inspection through the technique of watermarking.
Fig. 1. Overview of reconfigurable deep (binary) neural network. Rearranging the same network parameters (i.e., weight and activation) can switch to a network task from, e.g., natural image classification on CIFAR10 to digit recognition on MNIST. Notably, the parameter rearrangement can be individually applied to each layer. The bidirectional arrow means that the any two tasks can be flexibly reconfigured from one to the other.

An alternative is to leverage obfuscation and locking to further enhance the security of model IP [21], [29]. Xu et al. [29] structurally obfuscated the network architecture to prevent architecture steal. Instead of obfuscating the model architecture, Chakraborty et al. [21] obfuscated parameters of the model, which can only exhibit high inference performance unless an end user has legitimate access to unlock it. Without knowledge of the key being fed into the DNN, any unauthorized usage of the locked model shows a substantial accuracy drop, which is useless for the attacker.

It is recognized that our RBNN naturally enables the model locking/obfuscating property by associating the reconfiguration with a key, thus providing a very useful security benefit to protect the model IP.

D. Backdoor Attack by Abusing Multitask Learning

A backdoor attack introduces malicious behavior or tasks besides the main task for which the model is trained [23], [30]. In other words, a backdoored model essentially has multiple tasks: a main task and one or more backdoored tasks [30], [31], [32], [33]. The model trained for two tasks will exhibit both normal and backdoor behaviors, where the backdoor behavior is activated by the so-called trigger—a secret known only to the adversary. While previous backdoored tasks share the representation with the main task, TrojanNet [32] is a new backdoor technique for full-precision models rather than BNN that abuses the multitask learning property to make each backdoored task independent from the main task.

As summarized by Gao et al. [30], the backdoor can also be used for legitimate purposes such as watermarking for model IP protection [18], acting as a honey-pot for trapping and catching adversarial examples [34], and verification for data deletion requested by users who have contributed the data [35]. Our proposed RBNN employs the inherent multitask property abused by the backdoor attack for a legitimate use for resource-constrained IoT devices, e.g., improving memory efficiency of BNN-based models by embedding multiple tasks into a BNN model.

III. RECONFIGURABLE DEEP BNN

A. Overview

Fig. 1 presents an overview of our proposed reconfigurable deep BNN. By rearranging the order of network parameters, the same network with the same parameters executes completely different tasks. This requires that multitasks be trained upon distinct datasets: each dataset corresponding to a specific task. It is noted that the multitask training in RBNN does not learn shared representation among similar tasks as previously reported [22], rather, it learns different representations specific to each task. Nonetheless, both cases leverage the high capacity or redundancy of the DNN. Below we elaborate on how to realize RBNN through binarization.

B. Realization

1) Multitask Training: The key insight of RBNN is to utilize the multitask learning capability of DNN. Thus, different tasks \( \{T_1, T_2, \ldots, T_M\} \) share the same parameter set, including weights and biases. For convenience, we use \( T_m \) \( (m \in \{1, \ldots, M\}) \) to denote a network task. As illustrated in Fig. 1, each \( T_m \) can be immediately switched on by reconfiguring the network parameter order (i.e., parameter permutation).

Suppose that a dataset \( \mathcal{D}_m \) is used to train \( T_m \) and \( (x^m_i, y^m_i) \) denotes the \( i \)th sample from \( \mathcal{D}_m \). The overall loss \( L \) [see loss description in (1)] of the multitask training is the summation of individual task losses, which can be expressed as

\[
L = \sum_{m=1}^{M} \frac{L_{T_m}(T_m(x^m_i), y^m_i)}{L_{T_m}} = \frac{L_{T_1}(T_1(x^1_i), y^1_i)}{L_{T_1}} + \cdots + \frac{L_{T_M}(T_M(x^M_i), y^M_i)}{L_{T_M}}.
\]  

(3)
The overall loss $L$ can be optimized with gradient descent on $w_{T_m}$ and its gradient is given by

$$\frac{\partial L}{\partial w} = \sum_{m=1}^{M} \frac{\partial L_{T_m}}{\partial w_{T_m}}$$

(4)

The crucial step is to reconfigure the $w$ to fit $T_m$ and obtain $L_{T_m}$. This can be achieved by permutation, for example, $w_{T_2} \leftarrow \text{Reconfig}(w_{T_1})$ by reconfiguring the same parameters through permutation. In principle, we can train an arbitrary number of distinct tasks by the parameter permutations. In (3), the importance of each task is equal, thus the regularization is the same. Alternatively, we can use a distinct regularization factor for each $L_{T_m}$ to reflect the distinct importance of $T_m$ if needed.

Intuitively, we can store the permutation or the corresponding mask that corresponds to a task to facilitate the reconfiguration of Reconfig. However, this could introduce a non-negligible memory overhead. To address this issue, we customize the approach from [32] and generate the Reconfig in the RBNN. Specifically, the tasks are reconfigured through distinct keys: each task is bound up with a specific key (i.e., $k_m$). Here, $k_m$ is used as the initial seed fed into a recursive function, preferably a pseudorandom number generator (PRNG) $H$ albeit not a must, to produce recursive seeds that determine the permutation per subsequently network layer. Leveraging a recursive function, specifically $H$, could ensure the recursive seed-determined permutation per network layer is preferably distinct, thus a different reconfiguration even per layer. We have validated that our RBNN implementation is insensitive to the specific recursive function implementation, regardless of PRNG based or not, though the former is preferable.

In this way, Reconfig is associated with $k_m$. As such, $w_{T_m} \leftarrow \text{Reconfig}(w, k_m)$. A byproduct is that $w_{T_m}$ can only be configured correctly with a valid $k_m$. If an invalid key is applied, the parameters in the network will be wrongly permuted, causing the neural network to malfunction. To this end, the reconfiguration with valid keys issued from the model provider can provide IP protection to the network (see details in Section V).

As in (3), reconfigurable training requires training multiple different datasets in the same neural network. After training one batch of one dataset, the RBNN trains a data batch from the next task’s dataset in the same epoch. More specifically, the RBNN interleaves data from multiple tasks during learning to jointly optimize the model weights as a way to avoid catastrophic forgetting that could occur when a following task is trained after the previous task is completely trained [36]. When training alternates with dataset batches, $k_m$ is reconfigured to feed into $H$ and transform $w$ to $w_{T_m}$ so that $w_{T_m}$ corresponds to $D_m$ for $T_m$.

2) RBNN Training: When training an RBNN model, the backpropagation algorithm based on gradient descent can be used directly to update the parameters [10], [16]. However, the binarization function such as the sign function in (2) is usually not differentiable. Even worse, the derivative value of part of the function disappears, and the derivative of the function is almost 0 everywhere, which is obviously incompatible with backward propagation.

Thus, the original gradient descent-based backward propagation cannot be immediately used for updating the binarized weights. We solve this issue by using a pass-through estimator (STE) in backward propagation [37]. The derivative of STE can also be represented as the propagation gradient through the hard tanh, which is defined as

$$\text{Htanh}(x) = \max(-1, \min(1, x))$$

(5)

By applying the hard tanh, Htanh, activation function, the RBNN model can now be directly trained using the gradient descent that is the same when training the full-precision models. Nonetheless, if the absolute value of the full-precision activation is larger than 1, the Htanh$(x)$ cannot be updated in backward propagation. Therefore, the identity function (i.e., $f(x) = x$) is also chosen to approximate the derivative of the sign function in practice.

IV. EXPERIMENTS

The experiments in this section are designed mainly to answer the question (Q1) listed in Section I. We consider seven common datasets corresponding to six computer vision domain tasks and one audio domain task. The number of classes for each task ranges from 2 to 100. We use three common but different model architectures for a comprehensive evaluation.

A. Setup

1) Datasets: There are six image datasets under the vision domain and one speech recognition dataset under the audio domain, as summarized in Table I. We then demonstrate that those different domain tasks can be reconfigured as long as the same model architecture can handle these tasks.

**MNIST/Fashion-MNIST:** The MNIST has handwritten digits of ten classes by different people [38]. The Fashion-MNIST dataset [39] is composed of ten types of fashion products. Both contain 60,000 samples for training and 10,000 samples for testing. Each sample image is a grayscale or single-channel image with the size of 28 × 28 pixels.

**SVHN:** This dataset is composed of street view house numbers from Google [40]. The training set contains 73,257 samples and the test set contains 26,032 samples.

Each grayscale image has a size of 32 × 32. Similar to MNIST, the SVHN is also a digital recognition dataset, but is more difficult to recognize because of natural scenes.

**CIFAR10/100:** Both CIFAR10 and CIFAR100 are natural color image datasets, with CIFAR10 and CIFAR100 having 10
and 100 classes, respectively [41]. The training set and the testing set for both CIFAR10 and CIFAR100 contain 50,000 and 10,000 images, respectively. The CIFAR100 is harder to recognize than CIFAR10 due to the increased number of classes and decreased number of training samples per class.

CelebA: This dataset contains large-scale face attributes with more than 200K celebrity images, each with 40 attribute annotations [42]. The images in this dataset cover large pose variations and background clutter. CelebA has large diversities, large quantities, and rich annotations, including 10177 identities, 202,599 face images, and five landmark locations, 40 binary attribute annotations per image. In our work, the training set and the testing set are divided according to the list_eval_partition.txt file in the dataset—we do not use the validation set in this file. The training and testing sets have 162,770 and 19,962 samples, respectively. We have selected the gender attribute as a binary classification problem to train the model.

Speech Command (SC): This audio dataset has spoken words designed to help train and evaluate keyword spotting systems [43]. It consists of 105,829 utterances of 35 words. Each utterance is stored as a 1-s (or less) WAVE format file, with the sample data encoded as linear 16-bit single-channel pulse-code modulation (PCM) values, at a 16-KHz rate. Note that WAVE format files with a duration of less than 1 s are not used by us—these samples are excluded. As a consequence, a total of 95,394 audio data samples are left and randomly selected. Note that 60% of the data is used as the training set, and the rest is the testing set. Before feeding into the model, the mel-frequency cepstral coefficients (MFCCs) are used to extract speech features and convert them into a frequency spectrum. The sampling rate is set to 6300 Hz in order to convert the audio data format to $32 \times 32 \times 1$, and the mono data is copied to convert it into three channels.

2) Model Architecture: We consider binarizing three types of DNNs: ResNet18 [5], VGG [44], and ReActNet (using both ResNet14 and MobileNet as backbone, respectively) [45] for experimental validation. Note that ReActNet is a newly devised BNN framework to boost the BNN inference accuracy. It keeps the fully connected layers with full-precision representation by design, which we have followed. In addition, the ResNet backbone of ReActNet in our work is ResNet14, where the last block of ResNet18 is removed. This ResNet14 setting along with MobileNet backbone use are to test whether the smaller model’s redundancy still supports the reconfiguration without a drop in accuracy when multiple tasks are simultaneously learned.

For the VGG model, the architecture is $\text{conv}(128) \rightarrow \text{conv}(256) \rightarrow \text{conv}(512) \rightarrow \text{fc}(1024) \rightarrow \text{fc}(1024) \rightarrow \text{fc}$ (number of classes) where $\text{conv}(-)$ is a convolutional layer with $3 \times 3$ kernel size, and $\text{fc}(-)$ is a fully connected layer. For the ResNet model, the specific ResNet18 is used. For both VGG and ResNet models, all convolutional layers and fully connected layers are binarized, and a batch normalization layer is added before the activation layer and after the fully connected layer. For the ReActNet model, the ResNet14 and MobileNet are used as the basic skeleton of the network. The ReActNet by design reserves a fully connected layer to be represented with full-precision weights instead of being binarized. For all three models, the ADAM optimizer is used. The batch size is 256. We train all tasks with 100 epochs. The initial learning rate is $5 \times 10^{-3}$, which decreases to $1 \times 10^{-3}$ at the 40th epoch and $5 \times 10^{-4}$ at the 80th epoch.

We sequentially integrate six tasks under the image domain (MNIST, Fashion-MNIST, SVHN, CIFAR10, CelebA, and CIFAR100) to train each of the aforementioned model architectures to verify the RBNN performance (note that the RBNN is independent of the task adding sequence as detailed in Section VI-D). In addition, we have added one more audio task, SC, when training the RBNN with ReActNet. This is to demonstrate the practicality of co-learning tasks from different domains as long as the specific model is capable of handling those tasks. The switch of each seven tasks is controlled with a distinctive key. In this way, the specific task can be configured immediately by using its key during the inference phase.

3) Input Size Alignment: The image size of different classification tasks differs (see Table I). For all the model architectures studied in this work, the model accepted input size is $32 \times 32$. In this context, for the image size that is unequal to $32 \times 32$, we resize it to the model accepted input size. In addition, the model input image uses three channels by default. Therefore, for images, e.g., MNIST, SVHN, as well as the spectrum of SC, with a single channel, we have first transformed them into three channels before feeding them into the models.

B. Experimental Results

1) Baseline: To set a baseline, we represent the model parameters with full precision (float32) and 1 bit, respectively, and choose to train the CIFAR10 using all these model architectures. For the float32 model, we use typical training settings of Relu (BNN uses HTanh), ADAM optimizer, a batch size of 32 and a learning rate of $5 \times 10^{-4}$. The float32 model is trained with 100 epochs which is the same when training the BNN model. Notably, this typical setting is not aimed at achieving state-of-the-art (SOTA) performance [46], which will require special training settings, such as l2 regularization of the weights, warm-up, and cosine annealing recession of the learning rate and powerful data enhancement methods, or even delicate loss functions [46, 47, 48, 49].

A Comparison in Accuracy Between Full-Precision and 1-Bit Model: Results are detailed in Fig. 2. It is expected that the BNN model accuracy is lower than the full-precision model due to the tradeoff between accuracy and precision [25]. Notably, the focus of this work is not to optimize the BNN accuracy itself—for instance, one can exploit Neural Architecture Search in the binary domain [50] for doing so; instead we aim to integrate multiple BNN tasks by sharing the same model parameter set. Our concern is whether

\[2\text{The source code at https://github.com/liuzhechun/ReActNet. is adopted}\]
adding more tasks will greatly affect the accuracy of the co-learned individual task. Once the baseline of the BNN accuracy is optimized, our RBNN will naturally inherit the accuracy optimization benefit.

Model Size Comparison Between Full-Precision and 1 Bit Model: The number of parameters of VGG, ResNet18, and ReActNet (ResNet14 as backbone) is 14.03, 11.18, and 2.78 MiB, respectively. There is about $30 \times$ model size reduction as a result of using the BNN model in comparison to its full-precision model counterpart. For instance, the full-precision model size of the VGG is up to 53.56 MiB, while its BNN model size is only 1.74 MiB. This demonstrates the advantage of employing BNN when the storage memory is limited.

Accuracy Baseline of Individual Task With 1-Bit Model: The baseline accuracy of each task individually trained by each of the three models is summarized in Table II. The testing accuracy is evaluated after 100 epochs of training—further increase in the number of epochs does not result in noticeable improvement in the accuracy in our experiments. The accuracy reported in Table II serves as the baseline when compared with the accuracy of each task in the RBNN when multiple tasks are concurrently trained.

Here, we can see that the ReActNet has the best accuracy for the same task even though it is with the smallest model size. This is due to the fact that ReActNet is an optimized BNN framework that retains the full-precision representation for the fully connected layers to maintain the model accuracy, in contrast to the other two networks that use binary representation through the whole network.

2) RBNN Performance: Here, we sequentially add individual tasks one-by-one into the RBNN to see how and to which extent the number of tasks will affect the accuracy of individual task. Comprehensive results are detailed in Fig. 4 for VGG, ResNet18, ReActNet (ResNet14 as backbone), and ReActNet (MobileNet as backbone), respectively. Corresponding to Q1, we have the following observations.

Based on experiments with up to seven tasks, it is clear that multiple tasks can be embedded in the same BNN model, thus corroborating the RBNN. In addition, the accuracy of each task in the RBNN is similar to the trained alone task—this accuracy difference is less than 1% in most cases. This is also exemplified in Fig. 3 by the accuracy change of the CIFAR10 task (it is the first task added) when more tasks are sequentially integrated. In addition to accuracy at the end of the RBNN training shown in Fig. 4, the accuracy and loss changes per task during the RBNN training procedure are detailed in Fig. 5. This further affirms Q1, where each task is learned concurrently and converged as the number of epochs increases—no over- or under-fitting is observed. Essentially, the loss and accuracy curves are experimental evidence that the catastrophic forgetting has been properly resolved in RBNN when multiple distinct tasks are co-learned.

As for RBNN with a varying number of classes per task, it is still efficient. According to Table I, CelebA task has two classes, the SC task has 35 classes, the CIFAR100 task has 100 tasks, and all the rest four tasks have ten classes. Therefore, the number of classes for each task varies greatly from 2 to 100. Overall, these tasks can co-exist with negligible accuracy drop. The worst accuracy drop occurs when the CIFAR100 is
Fig. 4. Testing accuracy of each individual task as the number of tasks increases. (a) VGG. (b) ResNet18. (c) ReActNet (ResNet14 as backbone). (d) ReActNet (MobileNet as backbone).

Fig. 5. Loss and accuracy curves of each task during the RBNN training procedure. ReActNet with ResNet14 as backbone is used.

TABLE III
TESTING ACCURACY WHEN TWO TASKS ARE INTEGRATED UNDER VARYING QUANTIZATION WIDTH

| Model   | Task       | 1 bit Quant. Acc. (R+B) | 4-bit Quant. Acc. (R+B) | 8-bit Quant. Acc. (R+B) |
|---------|------------|-------------------------|-------------------------|-------------------------|
| ResNet14 | CIFAR10    | 69.6%; 84.4%            | 91.7%; 88.6%            | 96.7%; 95.9%            |
| ResNet14 | CIFAR100   | 54.7%                  | 55.5%                  | 57.6%                  |

(B:R): B means baseline accuracy when the task is individually trained; R means when multiple tasks are co-learned, standards for reconfiguration.

added, as it has the largest discrepancy with all other tasks in terms of the number of classes. For instance, in the ResNet18, the CIFAR10 accuracy drops by about 2.8% when the number of tasks increases from 5 to 6 (the sixth task is CIFAR100). At the meantime, Fashion-MNIST and SVHN see an accuracy drop of about 1%. Evaluations on VGG and ReActNet model architectures demonstrate a similar tendency. Notably, the accuracy drop of other tasks when CIFAR100 is added is minimal for the ReActNet, the potential reason lies on the fact that ReActNet 1) is delicately designed for BNN and 2) has a full-precision presentation of the fully connected layers. The latter could be the main reason affording a better compatibility of tasks with a varying number of classes.

As for integrating tasks from different domains, we have added the SC task as an audio domain task that is distinct from the image classification of all other six tasks. The results are shown in Fig. 4 using the ReActNet with ResNet14 and MobileNet as backbone, respectively. When SC is added, the accuracy drop of the other six image tasks is negligible, while the SC accuracy drop is about 1% and thus also unnoticeable. Therefore, it is reasonable to conclude that the RBNN is able to learn tasks across different domains as long as the same model architecture can be used to these domains. This appears to be promising in practice as using the same model architecture for cross-domain tasks is efficient. For instance, the transformer model that achieves significant success in natural language processing, recently has demonstrated great potential in image-domain tasks [52] and GAN-domain tasks [53].

Based on the experiments for three distinct model architectures, it is reasonable to conclude that the RBNN is generic to model architecture, regardless of whether the model itself is specifically designed for BNN, which provides an affirmative answer to Q1. Noting the VGG and ResNet are not delicate for BNN, while only the ReActNet is.
3) Increasing Quantization Width: Though our main focus is BNN, here we extend the reconfiguration to other quantization widths, in particular, with 4-bit and 8-bit quantization. We use ResNet (in particular, ResNet18) with two tasks of CIFAR10 and CIFAR100 for the extensive experiments in this part. We note that CIFAR100 has the largest number of classes. All other settings including learning rate and the number of epochs are similar to the one reported in Section IV-A2.

As can be seen from Table III, the higher quantization width the better the accuracy of each task. In our experiments, for both 4-bit and 8-bit quantizations, the CIFAR10 has been improved by more than 2%. For CIFAR100, it increases by more than 3%. In addition, we can see that the accuracy drop is minimized when a given task is co-learned in comparison with learned alone.

This further validates the generic nature of the proposed reconfiguration and shows that it is not solely limited to the concentrated BNN. Therefore, if the device has extra memory resources, then a slightly wider quantization approach can be used to improve the accuracy. Therefore, if the device has extra memory resources, then a slightly wider quantization approach can be used to improve the accuracy.

V. IP PROTECTION

It is recognized that the reconfiguration naturally enables model IP protection, e.g., allowing a user for trial and to only pay per the task needed (Q2). For instance, the default task is provided to the user for trial before payment. The trial task has partial functionality, e.g., classifying a limited number of classes. Once the user is satisfied with the trial and pays for the license, the model provider sends a key to the user, User_keyM, to reconfigure and thus unlock the task with a full functionality. Notably, we have shown that tasks with a varying number of classes can be effectively co-existed in RBNN. This scheme facilitates the user experience while protecting model provider IP. As for the user, she only needs to pay the task Tm that she is interested to save cost. Next, we propose a scheme to allow the user to pay not only per task but also per device.

Here, we propose to bind the key reconfiguring a different task with a unique device key that is extracted from the PUF [54, 55]. In general, the PUF is akin the fingerprint of the device, such a fingerprint can be converted into a usable cryptographic key, PUF_key. The PUF is a result of inevitable and uncontrollable manufacturing variations, thus preventing forging two identical PUF instances even under the same manufacturing process. Thus, the PUF_key is uniquely bound up with each device, which is hard to be physically cloned and replaced. Here, we can consider employing the intrinsic SRAM PUF to extract the PUF_key, since the SRAM is pervasively embedded within commodity electronics, requiring no additional hardware modification. The details of extracting PUF_key from (even low-end) commodity electronics in a lightweight manner can be referred to [56].

One prominent advantage of this PUF_key enabled reconfiguration is that the payment now can be made per device, and even per task. For the former, the model provider can use the same key_m to reconfigure the same task for avoiding training the RBNN per device. From the user’s perspective, the User_key_m is still different per device as the key_m is also a function of the PUF_key that is unknown, as illustrated in Fig. 6. Thus, the provider can issue different User_key_m to different devices given the same task. Here, we can simply set key_m = XOR(User_key_m, PUF_key)—admittedly, other means of deriving key_m upon User_key_m and PUF_key can be adopted.

As for the latter, the user can choose to only pay for a specific task Tm. If later another task is needed, the user can request from the model provider to issue another User_key, for the task of interest. In either case, the issued User_key_m can not be used on other devices even if every device has been provided all tasks by the model provider. However, without knowing the User_key_m that is specific to the device in-hand, the user cannot activate the task on the device, which protects the IP of the provider not only per device but also per task. At the same time, this allows the user to only pay for the task of interest on that device.

VI. COMPARISONS AND DISCUSSION

We compare the RBNN with other approaches and provide further discussions on such as its insensitivity to task adding order and adaption of activation functions.

A. Task Integration Comparison

As the RBNN is the first work demonstrating the feasibility of integrating multiple tasks into a single BNN model to reduce the DNN required memory footprint substantially, there is no other explicitly related work similar to our study.
TABLE IV
ACCURACY COMPARISON OF MULTITASK LEARNING BETWEEN MERGED DATASETS AND RBNN

| Order | Methods | CIFAR10<sup>1</sup> | MNIST<sup>2</sup> | Fashion – MNIST<sup>3</sup> | SVHN<sup>4</sup> | CelebA<sup>5</sup> | CIFAR100<sup>6</sup> |
|-------|---------|---------------------|------------------|------------------------|--------------|-----------------|------------------|
| -     | BNN(individually trained) | 86.8% | 99.3% | 94.1% | 95.3% | 97.1% | 55.9% |
| (1, 2, 3, 4) | Merged datasets | 85.4% | 99.0% | 93.3% | 93.7% | - | - |
|         | RBNN     | 86.0% | 99.1% | 93.7% | 94.6% | - | - |
| (1, 2, 3, 4, 5) | Merged datasets | 85.2% | 98.9% | 93.4% | 93.5% | 95.6% | - |
|         | RBNN     | 86.2% | 99.2% | 93.8% | 94.6% | 95.9% | - |
| (1, 2, 3, 4, 5, 6) | Merged datasets | 79.7% | 98.7% | 92.2% | 92.0% | 95.5% | 52.9% |
|         | RBNN     | 85.0% | 99.0% | 93.0% | 93.9% | 95.6% | 55.0% |

The superscript number in e.g., CIFAR10<sup>1</sup> means the number denotation of the task for an abbreviated reference.

Nonetheless, we have used the other means of integrating multiple tasks into the same model to serve as a multiple task integration baseline model, which generally merges these tasks into one and trains the model—thus, different tasks sharing similar latent representations. In this new baseline experiments, we merge 4–6 datasets and train the merged large dataset on ReActNet (ResNet14 as backbone) as the ReActNet is a delicate design for BNN. The accuracy of each merged task is (notably) smaller than that of RBNN (could be lower by 5.3%) as shown in Table IV. The difference in the inference accuracy between the merged tasks and the RBNN increases slightly when more datasets are merged, especially for datasets with feature similarity (we emphasize that CIFAR10 and CIFAR100 have no overlapped categories). More importantly, merging disparate data sets into a larger one and training with the same model architecture will no longer provide desirable IP protection as the RBNN does.

B. Continual Learning

Continual learning (CL), or long-life learning, may possibly an alternative that promises to reduce the memory requirement when it is incorporated into BNN, which we have not explored in this study. This is an open question for future work. Interestingly, we find that the techniques of isolating parameters per task exhibiting rigorous performance [57] in CL somehow share similar functionalities of the parameter permutation used in RBNN, where the tasks do not share the same latent representation to deal with forgetting. In addition, the insensitivity of task order in CL [57] aligns our validations of RBNN that is insensitive to the task order.

Nonetheless, the aims and learning processes of CL and RBNN are distinct. Note that CL has a (dynamic) sequential learning process to accumulate new knowledge when new data of the new task comes. However, the data of previous task(s)—the future task may also be unknown—is unavailable or only limited data samples are accessible. In other words, a major criterion of CL is the sequential nature of the learning process, where only a small portion of input data from one or few tasks are available at once [57]. In contrast, the RBNN has no such premise, as it trains multiple tasks with access to all training data at once during the co-learning. Once these multiple tasks are learned, they will not be updated by design since they will be sold as IP and deployed on IoT devices. In the CL, the semantics of the tasks play an important role in the amount of forgetting [58]. Nonetheless, catastrophic forgetting in CL is challenging but can be circumvented with special measures [57]. For RBNN, it does not need the semantics of the tasks (similarities/dissimilarities among tasks), as it integrates multiple models without sharing any latent representations among tasks. This enables the reconfigurability to be bound with a distinct key, thus protecting the model IP.

More specifically, the aims of RBNN are to reduce the memory and computation overhead when the deep learning model is deployed on mobile devices by switching inference tasks on demand. The RBNN has no any prior (semantic) requirement on the integrated tasks. The CL is a general approach to accumulate new knowledge sequentially, which can learn multiple tasks but appear to have stringent requirements on the semantics among tasks. The CL may be a promising alternative to reduce the memory and computation overhead by incorporating it with BNN. In addition, the RBNN is devised to provide IP protections by simply binding the task switch with a key, whereas the CL is unable to provide such a security gain, at least trivially.

C. Model Obfuscation Comparison

Chakraborty et al. [21] recently proposed the first work to leverage hardware as a root-of-trust to achieve IP security of DNN models, which is to obfuscate the model parameters in a way that the model is only correctly functioning on an authorized device with the unlocking key. From the security perspective of locking the DNN model (notably, security is one of two main benefits of our proposed RBNN), it is related to our work. From the security perspective, we differentiate our work from [21] in three main aspects.

1) Generally, in [21], the output of an internal neuron is now a function of both its original input and the hardware provided key—so bringing obfuscation for locking. The RBNN model locking technique is different in a way that we leverage parameter permutation in the RBNN training as a means of obfuscation—this is a byproduct of the RBNN to reduce memory footprint.

2) In contrast to the assumption in [21] that the key is securely stored in a secure storage in, e.g., trusted platform module (TPM), we do not require the secure key storage, as secure storage is unlikely available for a range of medium/low-end mobile devices. We propose to utilize the SRAM PUF that is pervasively used in
TABLE V
EFFECT OF TASK ADDING ORDER ON MODEL PERFORMANCE

| Order | CIFAR10 | MNIST  | Fashion-MNIST | SVHN   | CelebA  | CIFAR100  |
|-------|---------|--------|---------------|--------|---------|------------|
| (1, 2, 3, 4, 5, 6) | 85.0% | 99.0% | 93.0% | 93.9% | 95.6% | 55.0% |
| (6, 5, 4, 3, 2, 1) | 84.9% | 99.0% | 93.1% | 93.7% | 95.7% | 55.0% |

The superscript number in, e.g., CIFAR10 refers to the number of the task for an abbreviated reference.

TABLE VI
EFFECT OF ACTIVATION FUNCTION ON MODEL PERFORMANCE

| Activation function | CIFAR10 | MNIST | Fashion-MNIST | SVHN | CelebA | CIFAR100 |
|---------------------|---------|-------|---------------|------|--------|----------|
| softplus            | 84.2%   | 99.0% | 92.7%         | 93.7%| 95.5%  | 53.2%    |
| ReLU6               | 84.1%   | 98.8% | 92.7%         | 93.6%| 95.4%  | 52.9%    |
| HTanh               | 85.0%   | 99.0% | 93.0%         | 93.9%| 95.6%  | 55.0%    |

electronic devices to provide the unique key bound up with the hardware. This indicates that our solution can be applied to a wider range of electronic computing devices.

3) During the training of the locked model in [21], the key value must be known. In our case, we do not need this prerequisite. More specifically, an RBNN task-specific key is a function of User_key and a device-unique PUF_key. The key can be arbitrarily selected but kept the same (note it is hidden from the user) among devices. Then, the user key User_key is authorized according to the device PUF_key, so that the user key is ensured not user-specific but also device specific.

D. Task Adding Order

Based on (3) where subloss of each task has an independent addition relationship, RBNN is insensitive to the adding order in principle. In addition, further experimental results are shown in Table V with a reverse order, which does confirm the RBNN’s invariant to the task adding order. As we can clearly observe that the accuracy of each task before and after reordering are almost the same with negligible variations caused by rerunning the model inference. More generally, the dataset of each task is used to update the reconfigured weights corresponding to the task independently. As long as different tasks are alternately trained in the same order for each batch unit and the order can be randomly initialized, the same performance of the RBNN can be ensured.

E. Hard-to-Learn Task

In principle, the RBNN scales to more difficult tasks. If the hard-to-learn task’s input size is larger and the number of classes is higher than the tasks tested in this work, a more complex underlying RBNN model is required. In other words, the hard-to-learn task itself could require a deeper model with larger input size to gain a better accuracy. As a consequence, the complexity of the underlying RBNN model should also be increased as a prerequisite.

F. Activation Function

RBNN by design is independent of the activation function, such as HTanh, softplus, and Relu. However, HTanh is more suitable for BNN since it naturally limits the activation output to between -1 and 1. This is the reason we chose to use HTanh in our RBNN realization as in (5). We now evaluate the RBNN performance when activation functions other than HTanh are adopted. Experiments are implemented on the ReActNet model (ResNet14 as backbone), where six subtasks were co-trained by the RBNN. The results are summarized in Table VI. The inference accuracy for each subtask except CIFAR100 was roughly equivalent when using different activation functions—HTanh is slightly better. For CIFAR100, HTanh (55%) is relatively higher than softplus (53.2%) and Relu (using ReLU6) (52.9%). A possible reason for this is that neither Relu nor softplus support negative values (e.g., Relu is truncated to zero—softplus is a smoothed version of Relu).

G. Limitations

The RBNN reconfiguration on-device is mainly determined by feeding a task-specific key, with key a function of User_key and a device-unique PUF_key. As the User_key is provided to the user once the user buys the IP, the PUF_key requires online derivation that can dominate the reconfiguration overhead. Based on a recent work [59], it has been shown that the pervasively electronic commodities integrated intrinsic SRAM PUF-based PUF_key can be securely provided with substantially reduced overhead. Specifically, it requires 44 082 clock cycles for a 128-bit PUF_key derivation when the realization is performed on a low-end nRF52832 chip—a popular RF-enabled MCU. Such small overhead appears not a concern for mobile devices or medium-end IoT devices that the RBNN target.

Nonetheless, as this work mainly focuses on proposing and validating a new paradigm for reducing the DNN model’s memory volume requirement by the RBNN and a framework to protect the DNN model IP naturally enabled by the RBNN, the overhead of end-to-end RBNN realization on a dedicated hardware or a microcontroller is not specifically evaluated.
This is a limitation of the present study, which is an open question for future work.

VII. CONCLUSION

We have proposed RBNN to further improve the memory utilization for resource-constrained IoT devices by a factor of $M$, with $M$ as the number of tasks concurrently embedded. Note that RBNN is built upon the ultimate binary quantization of DNN—the reconfigurable paradigm indeed does apply to non-BNNs. Comprehensive experiments validate the practicability and effectiveness of the RBNN by: 1) up to seven tasks trained over each of three common model architectures; 2) tasks with a varying number of classes; and 3) tasks spanning distinct domains. More specifically, the task accuracy in the RBNN has no or only negligible degradation in comparison with standalone trained BNN tasks. In addition, the RBNN naturally enables novel IP protection for the model provider. The IP of each task integrated within RBNN can even be issued per user as well as per device, which greatly reduces the IP purchase for consumers while preserving the commercial benefits for the model provider.

REFERENCES

[1] A. Boulemtafes, A. Derhab, and Y. Challal, “A review of privacy-preserving techniques for deep learning,” Neurocomputing, vol. 384, pp. 21–35, Apr. 2020.
[2] S. Kundu, Q. Sun, Y. Fu, M. Pedram, and P. Beerel, “Analyzing the confidentiality of indistinguishable teachers in knowledge distillation,” in Proc. Adv. Neural Inf. Process. Syst., vol. 34, 2021, pp. 9181–9192.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” Commun. ACM, vol. 60, no. 6, pp. 84–90, 2017.
[4] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size,” 2016, arXiv:1602.07360.
[5] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.
[6] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, “GhostNet: More features from cheap operations,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2020, pp. 1577–1586.
[7] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 6848–6856.
[8] R. David et al., “TensorFlow lite micro: Embedded machine learning on TinyML systems,” 2020, arXiv:2010.08678.
[9] Tensorflow Lite. Oct. 2021. [Online]. Available: https://www.tensorflow.org/lite.
[10] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 29, 2016, pp. 4107–4115.
[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: Imagenet classification using binary convolutional neural networks,” in Proc. Eur. Conf. Comput. Vis., 2016, pp. 525–542.
[12] A. Bulat and G. Tzimiropoulos, “XNOR-Net++: Improved binary neural networks,” in Proc. Brit. Mach. Vis. Conf. (BMVC), 2019, pp. 62. [Online]. Available: https://arxiv.org/abs/1909.13863.
[13] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, “Training binary neural networks with real-to-binary convolutions,” in Proc. Int. Conf. Learn. Represent. (ICLR), 2020. [Online]. Available: https://openreview.net/forum?id=BJg4NgBKvH.
[14] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, “YodaNN: An architecture for ultralow power binary-weight CNN acceleration,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 37, no. 1, pp. 48–60, Jan. 2018.
[39] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms,” 2017, arXiv:1708.07747.

[40] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading Digits in Natural Images With Unsupervised Feature Learning,” 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

[41] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.

[42] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep learning face attributes in the wild,” in Proc. Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 3730–3738.

[43] P. Warden, “Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition.” Apr. 2018. [Online]. Available: https://ai.google/research/pubs/pub43373.

[44] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556.

[45] R. Zheng, Z. Yu, Y. Zhang, C. Ding, H. V. Cheng, and L. Liu, “Learning class unique features in fine-grained visual classification,” in Proc. Eur. Conf. Comput. Vis., 2020, pp. 143–159.

[46] C. Gu, W. Liu, Y. Cui, N. Hanley, M. O’Neill, and F. Lombardi, “A binary architecture search,” in Proc. Eur. Conf. Comput. Vis. (ECCV), 2020, pp. 309–325. [Online]. Available: https://arxiv.org/abs/2003.01711.

[47] L. Geiger and P. Team, “Larq: An open-source library for training Binarized neural networks,” J. Open Source Softw., vol. 5, no. 45, p. 1746, Jan. 2020. [Online]. Available: https://doi.org/10.21105/joss.01746.

[48] N. Parmar et al., “Image transformer,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 4055–4064. [Online]. Available: https://proceedings.mlr.press/v80/parmar18a.html.

[49] H. Zhang, I. Goodfellow, D. N. Metaxas, and A. Odena, “Self-attention generative adversarial networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 7354–7363.

[50] Y. Gao, S. F. Al-Sarawi, and D. Abbott, “Physical unclonable functions,” Nat. Electron., vol. 3, no. 2, pp. 81–91, 2020.

[51] C. Gu, W. Liu, Y. Cui, N. Hanley, M. O’Neill, and F. Lombardi, “A flip-flop based arbiter physical unclonable function (APUF) design with high entropy and uniqueness for FPGA implementation,” IEEE Trans. Emerg. Topics Comput., vol. 9, no. 4, pp. 1853–1866, Oct.–Dec. 2021, doi: 10.1109/TSMC.2020.3002566.

[52] Y. Gao, Y. Su, S. Nepal, and D. C. Ranasinghe, “NoisFre: Noise-tolerant memory fingerprints from commodity devices for security functions,” 2021, arXiv:2109.02942.

Huming Qiu received the bachelor’s degree from Southeast University, Nanjing, China, in 2020. He is currently pursuing the master’s degree with the Nanjing University of Science and Technology, Nanjing.

Huming Qiu’s research interests include AI security and privacy.

Hua Ma is currently pursuing the Ph.D. degree with the School of Electrical and Electronic Engineering, The University of Adelaide, Adelaide, SA, Australia, under the supervision of S. F. Al-Sarawi and D. Abbott. Her research interests include machine learning security and optimization.

Zhi Zhang received the bachelor’s degree from Sichuan University, Chengdu, China, in 2011, the master’s degree from Peking University, Beijing, China, in 2014, and the Ph.D. degree in computer science from the University of New South Wales, Sydney, NSW, Australia, in 2021.

He is a Research Scientist with Data61, CSIRO, Sydney. His research interests are in the areas of system security, rowhammer, and adversarial artificial intelligence.

Yansong Gao (Member, IEEE) received the bachelor’s degree from the Henan University of Science and Technology, Luoyang, China, in 2008, the M.Sc. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2013, and the Ph.D. degree from The University of Adelaide, Adelaide, SA, Australia, in 2017.

He was with Data61, CSIRO, Sydney, NSW, Australia, as a Postdoctoral Research Fellow. He is with the Nanjing University of Science and Technology, Nanjing, China, as an Associate Professor. His current research interests are AI security and privacy, hardware security, and system security.

Yifeng Zheng received the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2019.

He is an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, China. He worked as a Postdoctoral Fellow with the Commonwealth Scientific and Industrial Research Organization, Sydney, NSW, Australia, and the City University of Hong Kong. His work has appeared in prestigious conferences, such as ESORICS, DSN, ACM AsiaCCS, IEEE PERCOM, IEEE INFOCOM, and IEEE ICDCS, as well as journals, such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON MULTIMEDIA, and IEEE TRANSACTIONS ON CLOUD COMPUTING. His current research interests are focused on security and privacy related to cloud computing, IoT, machine learning, and multimedia.

Dr. Zheng received the Best Paper Award in the European Symposium on Research in Computer Security 2021.

Anmin Fu received the Ph.D. degree in information security from Xidian University, Xi’an, China, in 2011.

From 2017 to 2018, he was a Visiting Research Fellow with the University of Wollongong, Wollongong, NSW, Australia. He is currently a Professor with the Nanjing University of Science and Technology, Nanjing, China. He has published more than 80 technical papers, including international journals and conferences, such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE INTERNET OF THINGS JOURNAL, COMPUTERS & SECURITY, IEEE ICC, IEEE GLOBECOM, and ACISP. His research interests include IoT security, cloud computing security, and privacy preserving.
Pan Zhou (Senior Member IEEE) received the B.S. degree (in the Advanced Class) from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2006, the M.S. degree from the Department of Electronics and Information Engineering, HUST, in 2008, and the Ph.D. degree from the School of Electrical and Computer Engineering, Georgia Institute of Technology (Georgia Tech), Atlanta, GA, USA, in 2011.

He is currently a Full Professor and a Ph.D. Advisor with the Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, HUST. His current research interests include security and privacy, big data analytics, machine learning, and information networks.

Prof. Zhou received the Rising Star in Science and Technology of HUST in 2017, and the Best Scientific Paper Award in the 25th International Conference on Pattern Recognition 2020. He is currently an Associate Editor of IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING.

Derek Abbott (Fellow IEEE) was born in South Kensington, London, U.K. He received the B.Sc. degree (Hons.) in physics from Loughborough University, Loughborough, U.K., in 1982, and the Ph.D. degree in electrical and electronic engineering from The University of Adelaide, Adelaide, SA, Australia, in 1997, under the supervision of K. Eshraghian and B. R. Davis.

His research interests include the areas of multidisciplinary physics and electronic engineering applied to complex systems. His research programs span a number of areas of stochastics, game theory, photonics, energy policy, biomedical engineering, and computational neuroscience.

Dr. Abbott received a number of awards, including the South Australian Tall Poppy Award for Science in 2004, the Australian Research Council Future Fellowship in 2012, the David Dewhurst Medal in 2015, the Barry Inglis Medal in 2018, and the M. A. Sargent Medal for eminence in engineering in 2019. He has served as an Editor and/or a Guest Editor for a number of journals, including the IEEE JOURNAL OF SOLID-STATE CIRCUITS, Journal of Optics B, Chaos, Fluctuation and Noise Letters, PROCEEDINGS OF THE IEEE, and is currently on the editorial boards of PROCEEDINGS OF THE IEEE, and is currently on the editorial boards of Scientific Reports (Nature), Royal Society OS, Frontiers in Physics, PNAS Nexus, and IEEE ACCESS. He serves on the IEEE Publication Services and Products Board and is the current Editor-in-Chief of IEEE ACCESS. He is a Fellow of the Institute of Physics, U.K., and an Honorary Fellow of Engineers Australia.

Said F. Al-Sarawi (Senior Member, IEEE) received the general certificate in marine radio communication and the B.Eng. degree (First-Class Hons.) in marine electronics and communication from the Arab Academy for Science and Technology, Alexandria, Egypt, in 1987 and 1990, respectively, and the Ph.D. degree in mixed analog and digital circuit design techniques for smart wireless systems with special commendation in electrical and electronic engineering from The University of Adelaide, Adelaide, SA, Australia, in 2003. He also received the Graduate Certificate in education (higher education) from The University of Adelaide in 2006.

He is currently the Director of the Centre for Biomedical Engineering and a Founding Member of the Education Research Group of Adelaide, The University of Adelaide. His research interests include design techniques for mixed-signal systems in complementary metal–oxide–semiconductor and optoelectronic technologies for high-performance radio transceivers, low-power and low-voltage radio-frequency identification systems, data converters, mixed-signal design, and microelectromechanical systems for biomedical applications. His current educational research is focused on innovative teaching techniques for engineering education, research skill development, and factors affecting students evaluations of courses in different disciplines.

Dr. Al-Sarawi was awarded The University of Adelaide Alumni Postgraduate Medal (formerly, Culross Prize) for outstanding academic merit at the postgraduate level. While pursuing his Ph.D., he won the Commonwealth Postgraduate Research Award (Industry).