Thwarting Replication Attack against Memristor-based Neuromorphic Computing System

Chaofei Yang, Beiye Liu, Hai (Helen) Li, *Fellow, IEEE,* Yiran Chen, *Fellow, IEEE,* Mark Barnell, *Member, IEEE,* Qing Wu, *Member, IEEE,* Wujie Wen, *Member, IEEE,* and Jeyavijayan Rajendran, *Member, IEEE*

***Abstract*— Neuromorphic architectures are widely used in many applications for advanced data processing and often implement proprietary algorithms. However, in an adversarial scenario, such systems may face elaborate security attacks including learning attack. In this work, we prevent an attacker with physical access from learning the proprietary algorithm implemented by the neuromorphic hardware. For this purpose, we leverage the obsolescence effect in memristors to judiciously reduce the accuracy of outputs for any unauthorized user. For a legitimate user, we regulate the obsolescence effect, thereby maintaining the accuracy of outputs in a suitable range. We extensively examine the feasibility of our proposed method with four datasets. We experiment under different settings such as activation functions and constraints such as process variations, and estimate the calibration overhead. The security vs. cost and performance vs. resistance range trade-offs for different applications are also analyzed. We then prove that the defense is still valid even if the attacker has the prior knowledge of the defense mechanism. Overall, our methodology is compatible with mainstream classification applications, memristor devices, and security and performance constraints.**

***Index Terms*—Neuromorphic computing, learning attack, security, memristor, obsolescence effect.**

I. Introduction

A. Motivation

On one hand, machine learning has been widely used in data processing applications to help users understand the underlying property of the data [1]. As a popular type of machine learning model, neural network [2] processes input data by multiplying them with layers of weighted connections. Many embedded hardware engines, including FPGA and System-on-Chip (SoC), have been developed to implement neural networks with high speed and efficiency, e.g., Qualcomm's cognitive computing platform [3].

On the other hand, memristor has been discovered as a device whose resistance depends on the historical profile of the voltage applied on it. The similarity between the programmable resistance state of memristors and the variable weight connection in neural networks simplifies the structure of circuit realization of a neural network. The compact structure, high energy-efficiency, and low power consumption of memristor-based learning systems greatly improve the data scale and computation capacity of learning applications in embedded systems [4].

Running learning models on an embedded device, though advantageous because of reduced processing times and high energy-efficiency, introduces security challenges. The learning model is exposed to the risk of being attacked by malicious users who have physical access to the device. Consider the following scenario: Assuming there is a drone carrying an image processing system, which is being used for its navigation and guidance systems. This system implements the proprietary learning algorithms on a memristor-based neuromorphic computing system (MNCS). If the drone is captured by an unauthorized third party, say, an attacker, he/she may apply inputs to the system, observe the outputs, and "learn" the proprietary algorithm implemented by the system [5]. Consequently, they can design a pirated system.

B. This Work

In this paper, we demonstrate how an attacker can learn and replicate the proprietary algorithm. Our analysis is independent of the learning model (e.g., support vector machine (SVM) [6], random forest [7], K-nearest neighbors [8]). We then propose a secure MNCS design to thwart such replication attacks by leveraging memristors obsolescence effect. While many architectures are designed to mitigate this usually undesirable phenomenon, we harness this effect to resist security threats. In the drone example, the embedded system will be periodically calibrated to ensure its usability. Once the drone is captured by attackers, the calibration mechanism will no longer work. The resistance of the memristors in the crossbars, i.e., the weights of the model, will gradually shift, thus aggravating the accuracy of the embedded system. Therefore, the attackers will not learn useful information for model replication, which guarantees the model privacy. Furthermore, our analysis demonstrates that the attacker will not achieve a reasonable accuracy of the replication model even under the best-case scenario.

A naïve implementation of this idea will incur performance overhead. Hence, we develop device-, circuit- and architectural-level techniques to balance security and performance overheads. Experimental results show that our design

C. Yang, H. Li, and Y. Chen are with Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA (e-mail: chaofei.yang@duke.edu, hai.li@duke.edu, yiran.chen@duke.edu).

B. Liu is with Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15260, USA (e-mail: bel34@pitt.edu).

M. Barnell and Q. Wu are with Air Force Research Laboratory, Rome, NY 13441, USA (e-mail: mark.barnell.1@us.af.mil, qing.wu.2@us.af.mil).

W. Wen is with Department of Electrical and Computer Engineering, Florida International University, Miami, FL 33199, USA (e-mail: wwen@fiu.edu).

J. Rajendran is with Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843, USA (e-mail: jv.rajendran@tamu.edu).

Manuscript received March 21, 2019.

provides excellent usability as well as resilience to replication attack of the MNCS, without increasing calibration overhead. Previously, memristor devices are used to build security primitives such as physical unclonable functions [9], public physical unclonable functions [10], and motifs to prevent side-channel attacks [11]. Here, we focus on using them to prevent replication attacks on MNCS, which none of these primitives can thwart.

II. Preliminary

A. Memristors

The resistance of a memristor can be programmed by applying appropriate current or voltage pulses. With proper combinations of programming voltage amplitude and duration, the resistance of the memristor can be programmed to an arbitrary state between the low resistance state (LRS) and the high resistance state (HRS) [12]. The resistance of a memristor can be read (sensed) by a small current or voltage pulse. However, for most types of memristors, even a small read signal can disturb the resistance of the memristor, since the only difference between the read and write operations is the amplitude and/or the duration of the applied signal.

B. Obsolescence Effect of Memristors

The resistance of a memristor gradually changes on applying voltage pulses, eventually leading to either the ON state or the OFF state. We call this effect as the obsolescence effect of a memristor, as the original resistance value "vanishes" on applying a voltage pulse. The obsolescence effect happens because of two phenomena: 1) the intrinsic retention property of the device [13] and 2) the read-induced change in resistance. The first type of resistance change is hard to control since it is related to the material relaxing mechanism. The read-induced change is depicted in [13]. A memristor is constantly stimulated by short minor voltage pulses, and its resistance change (reflected by the sensed current) is recorded for every input pulse. This experiment is designed to mimic the impact of the small sensing signal applied to the memristor during read operations. It shows that the resistance of the memristor keeps increasing with the stimulation. Therefore, the obsolescence rate (i.e., changing rate of its resistance) can be controlled by choosing the amplitude and duration of the sensing current/voltage. In general, the resistance (or conductance) change of a memristor is a continuous procedure that can be described as:

$$\Delta R = f(v, t). \tag{1}$$

Here, ΔR is the resistance change. v and t are the sensing voltages and operation time of the memristor, respectively.

C. Memristor-based Neuromorphic Computing Systems

In this paper, we define a neuromorphic computing system as the hardware specifically designed to accelerate neural networks or machine learning algorithms. We also constrain our research object to supervised learning systems. Several such systems have been proposed by different research groups: As two major examples, IBM recently released their SRAM based neural chip, namely, TrueNorth [14], and Micron demonstrated Automata Processor [15] based on CMOS technology. A simple neural network that can be directly mapped onto an MNCS can be represented by two layers of neurons are fully connected by one layer of synapses. The output neurons collect the information from the input neurons through a network of synaptic connections and process them with a transfer function. The synapses multiply the signal transferring on them with different synaptic weights. In general, the relationship between the value of the input vector $\boldsymbol{x}$, and the output vector $\boldsymbol{y}$ can be described by [16]:

$$\boldsymbol{y}_n = f(\boldsymbol{W}_{m\times n} \cdot \boldsymbol{x}_m). \tag{2}$$

Here, the connection weight matrix $\boldsymbol{W}_{m\times n}$ denotes the synaptic strengths between the two layers of neurons, n and m denotes the neuron number of current layer and previous layer. The matrix-vector multiplication in Eq. 2 is one of the fundamental operations in neural network and machine learning algorithms. Due to the structural similarity, memristor crossbars are time-efficient platforms to execute such matrix-vector multiplications [17]. The operation defined by Eq. 2 is the feedforward "evaluating" operation of a traditional neural network. During the evaluating process of an MNCS, $\boldsymbol{x}$ is represented as a vector of voltage signals applied to the word-lines (WLs) of the memristor crossbar while the bit-lines (BLs) are grounded. The current sensed from the bottom of each BL will be converted to output voltage vector $\boldsymbol{y}$ by a specially designed sensing circuit. Here the sensing circuit can be a CMOS analog module or a memristive device carrying the necessary transformation function. The matrix $\boldsymbol{W}_{m\times n}$ is often implemented by two memristor crossbars, which represent the positive and negative elements of $\boldsymbol{W}_{m\times n}$, respectively. "Training" on this system denotes the process of programming the memristors to the conductance states representing $\boldsymbol{W}_{m\times n}$. Open-loop and close-loop are two major training schemes. The former directly applies a programming pulse on the targeted memristor. The latter updates the $\boldsymbol{W}_{m\times n}$ iteratively based on the discrepancy between the generated and the expected outputs.

III. Thwarting Learning attacks

A. Target System

An MNCS consists of the following two proprietary information:

- *Training data* denotes the sample set used for training the MNCS. Each sample normally contains a vector of features and a label. The feature vector serves as the input of the learning model, and the label describes a property.
- *Learning model* denotes the model that has been trained for the proprietary application using the training data. It includes two parts: 1) the model info, say, the type (e.g., Hopfield or Naïve Bayes models) and the topology, and 2) the model parameters, *e.g.*, the weight on each synapse.

Without losing generality, we assume the function of the original learning model $g(\boldsymbol{w}, \boldsymbol{x})$ is data classification, which can be described as:

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/TCAD.2019.2937817, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

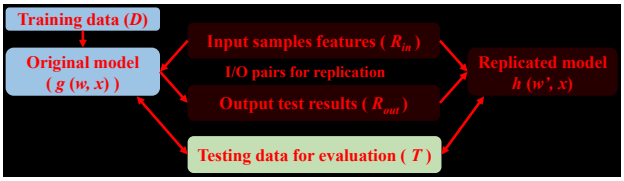


Fig. 1. Training and replication process of the learning model [18].

$$g(\boldsymbol{w}, \boldsymbol{x}) = \max_{y_i} p(y = y_i | \boldsymbol{w}, \boldsymbol{x}), i = 1, ..., n. \quad (3)$$

Here, $\boldsymbol{w}$ represents the parameters of the original model. $\boldsymbol{x}$ is the input vector of features. y_i is the i_{th} target class that a sample can be assigned to. The probability function $p(y = y_i|\boldsymbol{w}, \boldsymbol{x})$ is defined by the structure of the original model, e.g., a neural network. After the training completes, the original model $g(\boldsymbol{w}, \boldsymbol{x})$ is ready to classify new evaluating data.

B. Protocol

Fig. 1 shows a conceptual view of the concerned embedded system and its usage model. A proprietary (classification) algorithm is running on the hardware, e.g., an MNCS. The model is first trained for an application, and then the drone can submit the collected data for processing (evaluating), e.g., pattern recognition or classification.

The drone with the learning system executes the following protocol:

1) Initially, the MNCS is not trained, and hence it does not implement the proprietary algorithm.
2) The drone requests the base station for the training samples.
3) After authenticating the drone, the base station establishes a secure session with the drone using conventional cryptographic protocols.
4) The base station encrypts the training set and sends to the secure session.
5) The drone decrypts the encrypted training set and trains the MNCS to implement the proprietary algorithm.
6) The MNCS executes proprietary algorithm *N* times, after which the weights erase due to memristor's obsolescence effect.
7) After applying *N* I/O pairs, repeat steps (2) through (6).

C. Threat Model

We assume that the attacker has the following capabilities:

- The attacker can apply inputs, e.g., images, body data from patients, finger prints, to the originally trained model and obtain the corresponding outputs without any constraints, i.e., being granted with the same privilege as a normal user or being able to physically get access to it.
- The attacker does not have access to the original training set.
- The attacker has no knowledge about the parameters of the original model.
- An attacker can reverse engineering to understand the hardware implementation of the system.

The objective of the attacker is to replicate the function of the original model $g(\boldsymbol{w}, \boldsymbol{x})$ by constructing a new model $h(\boldsymbol{w'}, \boldsymbol{x})$, such that the $h(\boldsymbol{w'}, \boldsymbol{x}) = g(\boldsymbol{w}, \boldsymbol{x})$. To achieve this goal, an attacker can perform the following attacks:

1) **Eavesdropping attack**. An attacker can listen to the communication channel to obtain the training set. This attack is not possible, because the training set is encrypted and sent across the channel, as stated in Step (3) of the protocol.
2) **Spoofing attack**. An attacker can impersonate as a drone and request for the training set from the base station. This attack is not possible, because the base station authenticates the drone before sending the training set, as stated in Step (3) of the protocol.
3) **Probing attack**. An attacker can probe the memristors and can try to learn the stored weights [19]. Since the attacker already has the structure of the MNCS through reverse engineering, in addition to the weights, he/she can replicate the proprietary algorithm. This attack is not possible, because memristors are highly dense and can be compactly stacked in 3D structure, making them difficult to probe without physically damaging the neighborhood devices. Besides, countermeasures can be used to prevent probing attack [18].
4) **Chosen input attack**. An attacker can apply inputs of his/her choice, observe the corresponding outputs, and infer the weights. In this proposal, we focus on this attack and thwart it using the obsolescence effect of the memristor in the MNCS.

D. Chosen-input Attack

In this proposal, we use $\boldsymbol{D}_{in}$ to denote the inputs chosen by the attacker to the MNCS. $\boldsymbol{D}_{out}$ is the output of the MNCS. $[\boldsymbol{D}_{in}, \boldsymbol{D}_{out}]$ construct I/O pairs. Here the length of $\boldsymbol{D}_{in}$, i.e., the number of inputs chosen by the attacker and the length of Dout and $[\boldsymbol{D}_{in}, \boldsymbol{D}_{out}]$ are the same, say, m, which is decided by the attacker.

Since the attacker has no knowledge about the type of the original model, he/she needs to select a learning model as a starting point for *model replication*. After the I/O pairs are constructed and the replicated model type is selected, the attacker starts to use the I/O pairs to train the replicated model: because the I/O pairs are generated from the original model, the function of the replicate model will gradually approach to that of the original model.

Since the attacker does not know the model implemented by the MNCS, any arbitrary model may be selected. Besides the original model, (e.g., neural network [2]), we could also use other model (e.g., support vector machine [6]) as the replicated model to learn the function of the original model. In addition, it has been proved that although the selection of learning model is crucial for replication efficiency and accuracy, it is not necessary to select the same model type as the one of the original model [20].

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/TCAD.2019.2937817, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

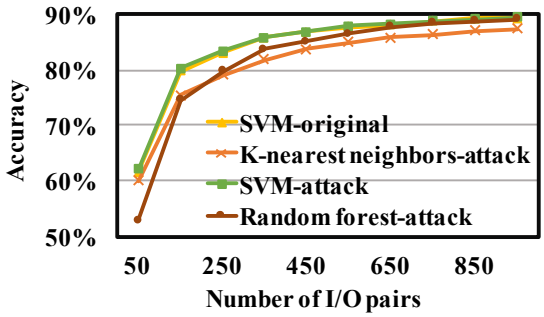


Fig. 2. Comparison of learning model between the original model and various other learning models and parameters.

Security Metric. The performance of the MNCS is evaluated by their accuracy, which is defined as:

$$accuracy = \frac{number\ of\ true-positives}{number\ of\ all\ evaluating\ samples}. \quad (4)$$

Here the number of true-positives is the number of predictions that match the ground-truth labels. In this proposal, we use accuracy as the security metric to quantify the effectiveness of our attack.

To demonstrate the effect of different learning models on accuracy, we use *MNIST* dataset as an example [21]. *MNIST* is a handwritten digit dataset, which is widely used in machine learning field and various image processing training. The system implements the target application using SVM [6]) model. Other candidate learning models include: SVM [6]), random forest [7], and K-nearest neighbors [8]. The attacker does not know which of the four learning model is being implemented in the system.

As we mentioned in Section III-B, the attack models take the I/O pairs generated from the original model as their training data. A normal model (e.g., a SVM model) trained by original training labels is also evaluated for comparison. Experimental results can be found in Fig.2. The replicated model based on SVM shows a rate of increase in accuracy w.r.t. I/O pairs similar to that of the original one. Even if the replication attack uses other learning models, the rate of increase in accuracy w.r.t. I/O pairs is similar. And, their accuracy's both approach to the one of the normal model (90%) after applying 900 I/O pairs.

This experiment shows that the model replication attack is feasible and even if the replication model is different from the original model (SVM, in this case), it still can achieve a good enough accuracy. Thus, the proposed defense mechanism should prevent the attacker from replicating the algorithm, irrespective of the underlying learning model.

IV. Secure MNCS Design

A. Device Level: Memristors

Among all the device candidates such as memristor, phase change memrory, and other non-volatile memories [22], we use memristor because of its following attractive properties:

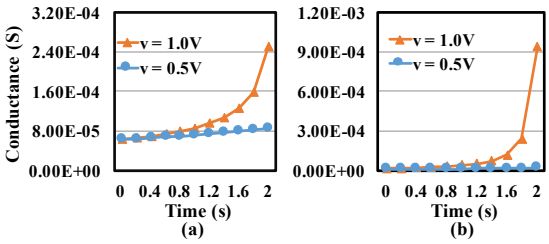


Fig. 3. Memristor model: (a) Conductance. (b) Derivative of Conductance.

1) Memristors are highly dense and can be stacked in 3D structure, which makes it extremely difficult for physical attacks.
2) Memristor is energy-efficient.
3) Memristor is programmable for online training.

In this paper, we adopt the memristor model from the work of Miao *et al.* [23]. The memristance can be expressed as:

$$M(\alpha) = \alpha \cdot R_L + (1-\alpha) \cdot R_H, \quad (5)$$

where α is the relative doping front position which ranges from 0 to 1. It can be obtained by solving the differential equation of velocity:

$$\alpha(t) = \frac{R_H - \sqrt{{R_H}^2 - 2 \cdot (R_H - R_L) \cdot (A+B)}}{R_H - R_L}, \quad (6)$$

where $A = \mu_v \cdot \frac{R_L}{h^2} \cdot \int_{t_0}^{t} V(t)\,dt$ and $B = R_H \cdot \alpha_0 + \frac{1}{2} \cdot (R_H - R_L) \cdot {\alpha_0}^2$. α_0 is the initial condition of α. Assume $t_0 = 0$, $\alpha_0 = 0.3$ and substitute A and B into α and then to $M(\alpha)$, we have:

$$M(t) = \sqrt{\begin{array}{r} {R_H}^2 - 2 \cdot (R_H - R_L) \cdot (\mu_v \cdot \frac{R_L}{h^2} \cdot v \cdot t \\ +0.255 \cdot R_H + 0.045 \cdot R_L) \end{array}}. \quad (7)$$

By using the ideal memristor parameters ($h = 50nm, R_H = 16k\Omega, R_L = 100\Omega, \mu_v = 10^{-14} m^2 S^{-1} V^{-1}$ [23]), we can have a simplified memristor model as (in conductance form):

$$G(t) = 3.5 \cdot 10^{-4} \cdot (32 - 15 \cdot v \cdot t)^{-0.5} \quad (8)$$

and its corresponding derivative:

$$\frac{dG(t)}{dt} = 2.7 \cdot 10^{-3} \cdot (32 - 15 \cdot v \cdot t)^{-1.5}. \quad (9)$$

The conductance change over time can be found in Fig. 3. Voltages of 1.0V and 0.5V are applied to the same memristor, respectively, in our experiments. We made a observation that the curve has a relatively flat portion, which we can utilize to achieve a linear degradation speed, e.g., when applying a small sensing voltage (e.g., 0.5V) or decreasing the duration of the applied voltage pulses.

Our selected model is a general memristor model which was originally proposed by HP Labs [24]. This model was also adopted in many prior works [25], [26]. There are indeed various implementations of memristors and thus many versions

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/TCAD.2019.2937817, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

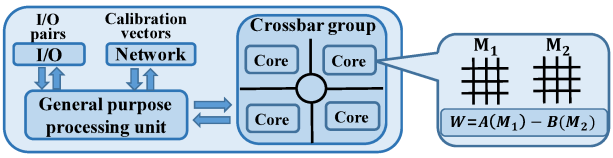


Fig. 4. Overview of MNCS structure.

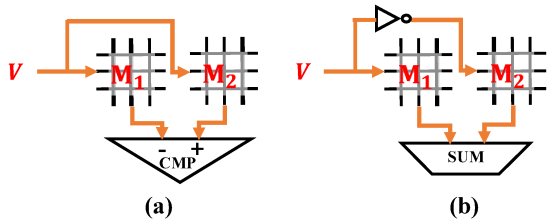


Fig. 5. Compare two proposed designs: (a) Naïve design with positive voltages applied on both crossbars. (b) Revised design with negative voltages applied on second crossbar.

of memristor models. However, our method should work on one specific model as long as there is a "linear" region in the g-t curve. Fortunately, this is true for most memristor models [27]. In practice, we can design the system such that the device works in the desired region. Later in Section V-E, the robustness analysis also indicates that our method will work under an imperfect situation.

B. Circuit Level

Fig. 4 depicts an overview of the MNCS structure. Since the elements in the weight matrix $\boldsymbol{W}$ of a neural network can be either positive or negative but the conductances of memristors can be only positive, we split $\boldsymbol{W}$ into two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as:

$$a_{ij} = \begin{cases} w_{ij}, & \text{if } w_{ij} > 0 \\ 0, & \text{if } w_{ij} <= 0, \end{cases} \tag{10}$$

$$b_{ij} = \begin{cases} 0, & \text{if } w_{ij} > 0 \\ w_{ij}, & \text{if } w_{ij} <= 0, \end{cases} \tag{11}$$

where w_{ij} denote the elements in $\boldsymbol{W}$.

Matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are represented using one memristor crossbar for each ($\boldsymbol{M_1}$ and $\boldsymbol{M_2}$, respectively) where the conductance of every memristor $g > 0$. Then we have:

$$\boldsymbol{y}_n = f(\boldsymbol{x}_m \cdot \boldsymbol{A}_{m\times n} - \boldsymbol{x}_m \cdot \boldsymbol{B}_{m\times n}). \tag{12}$$

For simplicity, here we assume the conductance change of memristors follows Eq. (8). In naïve design, same inputs are applied on both $\boldsymbol{A}$ and $\boldsymbol{B}$ during computation. If we use $\Delta\boldsymbol{A}$ and $\Delta\boldsymbol{B}$ to denote the weight change represented by $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. Then the change of $\boldsymbol{W}$ can be expressed as $\Delta\boldsymbol{W} = \Delta\boldsymbol{A} - \Delta\boldsymbol{B}$. According to Eq. (9)-(11), we have:

$$\Delta\boldsymbol{W} = sign(\boldsymbol{W}) \cdot dG(v,t) \cdot input. \tag{13}$$

where $sign()$ is sign function, $dG(v,t)$ can be changed by adjusting v and t, and $input$ is target application specific.

As the conductance of the crossbar changes upon applying an input, the accuracy of the system degrades over time, which means the function of the model implemented by the system is gradually changing. In order to control this property, we propose to apply random voltage pulses to all memristors for each I/O pair, so that the conductance can change linearly and evenly, across all memristors.

In order to guarantee stable, correct outputs for authenticated users, a calibration mechanism must be applied to such a system with forgetting property. A naïve way to calibrate is to refresh the crossbars with initial conductance states periodically, using the protocol listed in Section III-B. In order to boost calibration efficiency, we may adopt various techniques such as inline calibration [28].

When the accuracy of the MNCS is high, it offers a better service quality to normal users, but aids the attacker in learning the proprietary algorithm better because he can obtain outputs with higher accuracy. Degradation of accuracy prevents an attacker from accurately learning the model, but also reduces the accuracy for a normal user. In order to solve the above dilemma, in this paper, we propose to design a MNCS that has a very nonlinear degradation in accuracy (Fig. 5 (b)).

We use a classification application as an example. When we keep applying inputs, the classification accuracy of the MNCS degrades. The degradation rate of the accuracy is low initially so that it can provide accurate outputs to the authorized user. The degradation then sharply accelerates when the number of test operations exceeds a threshold to prevent the attacker from replicating the model by obtained sufficient number of I/O pairs.

We design the system by manipulating the input voltages applied on the memristor crossbar. In the naïve design, only positive inputs are applied to memristor crossbars $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, and the result from $\boldsymbol{M}_2$ will be deducted from the result of $\boldsymbol{M}_1$ in the post-processing logic. In such a design, the conductances of the memristors in both $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are changing in the same direction. In our revised design, as shown in Fig. 5, the inverted negative inputs are applied to $\boldsymbol{M}_2$ while the inputs to $\boldsymbol{M}_1$ still kept positive. The result of $\boldsymbol{M}_2$, hence, needs to be added on top of the result of $\boldsymbol{M}_1$. The conductances of the memristors in $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are now changing in the opposite directions, and the weight changing function will change from Eq. (13) to

$$\Delta\boldsymbol{W} = dG(v,t) \cdot input \tag{14}$$

Fig. 6 shows the weight change difference between the naïve and the revised designs. For the naïve design, because of the sign function, positive weights are changing to higher values and negative weights are changing to lower values. For the revised design, everything is changing to higher value. Changes in weight are represented by the shade in Fig. 6, which follows our expectation according to Eq. (13) and (14).

The revised design with negative inputs demonstrates a stronger nonlinearity than the naïve design when the number of test operations increases. Hence, we prefer to use the revised

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/TCAD.2019.2937817, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

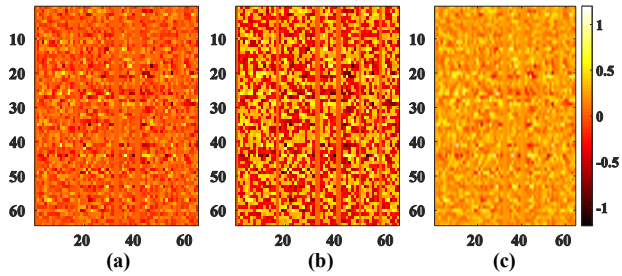


Fig. 6. Weight change in MNCS: (a) Original weight matrix. (b) Weight matrix in naïve design post-degradation. (c) Weight matrix after revised design post-degradation.

design over the naïve design. The comparison results will be shown in Section V.

C. Architectural level: Depth of Neural Networks

Increasing the depth of neural networks is another way to increase the nonlinearity of the MNCS. For example, a neural network has intrinsic nonlinearity that arises from the error diffusion across its layers. Considering a normal feedforward network with one hidden layer, we have:

$$\boldsymbol{y} = f_2(f_1(\boldsymbol{x} \cdot \boldsymbol{W}_1) \cdot \boldsymbol{W}_2)). \tag{15}$$

where $f_i (i = 1, 2)$ are the transfer functions, $\boldsymbol{W}_1$ is the weight matrix between input layer and hidden layer, $\boldsymbol{W}_2$ is the weight matrix between hidden layer and output layer, $\boldsymbol{x}$ is the input, and $\boldsymbol{y}$ is the output. Hence, the partial derivative of $\boldsymbol{y}$ respect to $\boldsymbol{W}_1$ is:

$$\frac{d\boldsymbol{y}}{d\boldsymbol{W}_1} = \frac{df_2}{df_1} \cdot \frac{df_1}{d\boldsymbol{W}_1} = f_2'(f_1(\boldsymbol{x} \cdot \boldsymbol{W}_1) \cdot \boldsymbol{W}_2) \cdot \boldsymbol{W}_2 \cdot f_1'(\boldsymbol{x} \cdot \boldsymbol{W}_1) \cdot \boldsymbol{W}_1. \tag{16}$$

For simplicity, we assume the second transfer function f_2 is a linear function. Then the derivative can be re-written as:

$$\frac{d\boldsymbol{y}}{d\boldsymbol{W}_1} = \boldsymbol{W}_2 \cdot f_1'(\boldsymbol{x} \cdot \boldsymbol{W}_1) \cdot \boldsymbol{W}_1. \tag{17}$$

The partial derivative of $\frac{d\boldsymbol{y}}{d\boldsymbol{W}_1}$ respect to $\boldsymbol{W}_1$ has a high degree (larger than 1) because the transfer function is usually a nonlinear function, e.g., hyperbolic tangent, which explains why neural network has intrinsic nonlinearity property.

Similar to the back propagation method, the errors generated from the weight matrix at the first layer will pass onto all the subsequent layers, and affect the accuracy. The deeper the network is, the greater the influence introduced by the errors will be. Similar conclusion can be drawn for the matrices at other layers. By carefully selecting the depth of the neural network used in the original model, we are able to control the nonlinearity of the degradation curve of the accuracy (service quality).

D. New Challenge With Prior Knowledge on Defense Mechanism

To further evaluate the effectiveness of our defense mechanism, we endow the attacker stronger capability by letting he know our defense mechanism in advance. This assumption is practical, for that even if the attacker does not know in advance, he will be able to know at least the basics of the defense mechanism by collecting related information or learn from the results. This means that before the attacker sends inputs to our system, he already knows that this system is going to degrade and the resistance of memristors will obsolete according to input samples (voltage pulses). However, the attacker can not access to the inside of the system, which means he can not control the weights or change the structure. Otherwise, it should be belong to what has been discussed in Section III. The attacker can only play some tricks on the input dataset, which is determined by himself, to slow down the system degradation or at least to avoid the worst case. There may exist the best way to do so and we will discuss it later.

New attack model formulation. Here, we formulate the above assumption as our new attack model. In general, the attacker should obey the following claims based on our basic attack model discussed in Section III:

1) The attacker knows the defense mechanism (system degradation) prior to sending the inputs into the system.
2) The attacker can not change the weights or the structure of the system.
3) The attacker can manipulate the inputs when attacking the system.
4) The attacker can learn from the outputs to re-arrange the inputs.

Mechanisms of defending-aware attack. We summarized several mechanisms the attacker may adopt to manipulate the inputs as follows, and we will evaluate their effectiveness in Section V-G.

1) ***Random*** is a normal way of sending inputs to the system. By doing this, the attacker should shuffle the input samples in random order. The result usually varies in a small range due to the randomness, meanwhile, the system degrades at a moderate speed.
2) ***Sequence*** is to send the inputs in a sorted sequence, e.g., send all samples in the 1st class and then send samples in 2nd class, and so on. This technique brings the fastest system degradation due to the accumulation of obsolescence effect of similar memristor crossbar

Algorithm 1 Generate attack samples using *difference* technique

Initialize the network.
Randomly choose a starting input sample $\boldsymbol{x}_1$.
while not all input samples determined **do**
1) calculate the difference between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j, j > i$
2) find $\boldsymbol{x}_j$ with largest difference
3) swap $\boldsymbol{x}_{i+1}$ and $\boldsymbol{x}_j$
4) $i = i + 1$

end while

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/TCAD.2019.2937817, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

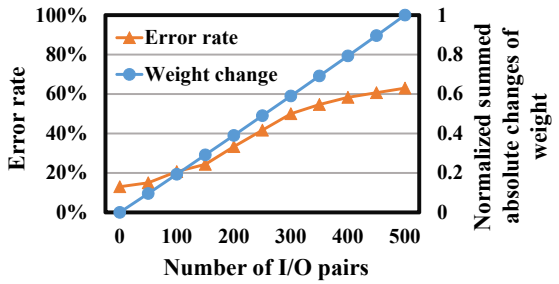


Fig. 7. The relation between weight change and system degradation.

regions.

3) ***Difference*** represents the technique to send the inputs with the largest difference. Here we use MSE to evaluate the difference, i.e., the difference between inputs $\boldsymbol{x}$ and $\boldsymbol{y}$ is $D(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$. Details of how to generate input samples using difference technique can be found in Algorithm 1. The sample with the largest difference from the prior sample will be selected as the next input sample.
4) ***Feedback*** is the technique manipulating input samples by taking advantage of the *softmax* outputs. Details of how to generate input samples using feedback technique can be found in Algorithm 2.

V. Results

In this section, we will demonstrate the effectiveness of our proposed MNCS design to defend replication attacks. In Section V-A, we will depict the obsolescence effect of memristor devices and its relationship with the system accuracy. The proposed revised design, which has nonlinear degradation, will be shown in Section V-B with the comparison to the naïve design that has linear degradation. In Section V-C, we will show the security advantages provided by the revised design. In our experiments, we choose two benchmarks from UCI machine learning repository [29]: Image Segmentation (*Image*), Steel Plates Faults (*Faults*), one benchmark from Scikit-learn [30]: Hand-writtern digits (*Digit*) and popular digit classification dataset *MNIST* [21]. All the details of the datasets are listed in Table I. These are all representative classification tasks that can be realized on memristor-based devices.

Algorithm 2 Generate attack samples using *feedback* technique

Initialize the network.
Randomly choose a starting input sample $\boldsymbol{x}_1$**.**
while not all input samples determined **do**
1) send $\boldsymbol{x}_i$ into the system and get *softmax* result;
2) find the least value of the *softmax* result, which represents class t;
3) randomly choose a sample $\boldsymbol{x}_j$ from class t and swap with $\boldsymbol{x}_{i+1}$;
4) $i = i + 1$.
end while

TABLE I
Summary of benchmarks.

	# Training	#Evaluation	#Attributes	#Class
Image	1500	810	19	7
Faults	1500	441	27	7
Digit	500	300	64	10
MNIST	50000	10000	784	10

A. Number of I/O pairs vs. Degradation in Accuracy

To evaluate the impact of obsolescence effect on the accuracy of the MNCS, we simulate the degradation in accuracy of MNCS when running different benchmarks. The memristor crossbar is configured to implement a neural network with two hidden layers for all the benchmarks. Each layer has 64 neurons. Compared with state-of-the-art models, such a shallow fully-connected network is relatively simple but serves as a good starting point. In Section V-B, we will also explore the influence of the depth of the network, indicating the proposed method will work on deeper networks. Furthermore, current mainstream deep networks are not practically feasible for memristor crossbars.

The simulation results are summarized in Fig. 7. Without loss of generality, we take *Digit* for example as the curves for all three benchmarks have very similar trend. X-axis denotes the number of I/O pairs. The left y-axis denotes the error rate of the system while the right y-axis denotes the normalized summed absolute changes of weights (NSCW) due to memristor obsolescence. We define the error rate and NSCW as:

$$error\ rate = \frac{1}{n}\sum_{i=1}^{n}\text{if}(\boldsymbol{y}_i == \boldsymbol{t}_i), \tag{18}$$

$$NSCW = \frac{\sum_{i,j}|w_{ij} - w'_{ij}|}{\sum_{i,j}|w_{ij}|}. \tag{19}$$

Here $\boldsymbol{y}_i$ denotes the classification result, $\boldsymbol{t}_i$ denotes the target result, n denotes the size of I/O pairs, and $\boldsymbol{w}_ij$ is the element of weight matrices. As the weight change increases, the error rate increases from less than 20% to over 60% gradually due to the obsolescence effect of the memristors.

The low error rate region at the beginning provides the usability for normal users. The average error rate within the first 50 I/O pairs only increases by less than 2%. More details on this will be provided in Section V-B.

B. Linear vs. Nonlinear Degradation

To provide better usability and increase protection against replication attack, we compare the naïve design and the revised design. The naïve design possesses a linear degradation curve, leading to slower degradation. An attacker can apply more I/O pairs with high accuracy and thus, learn the proprietary algorithm better. The revised design has a nonlinear degradation curve that degrades faster than the naïve design. The

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/TCAD.2019.2937817, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

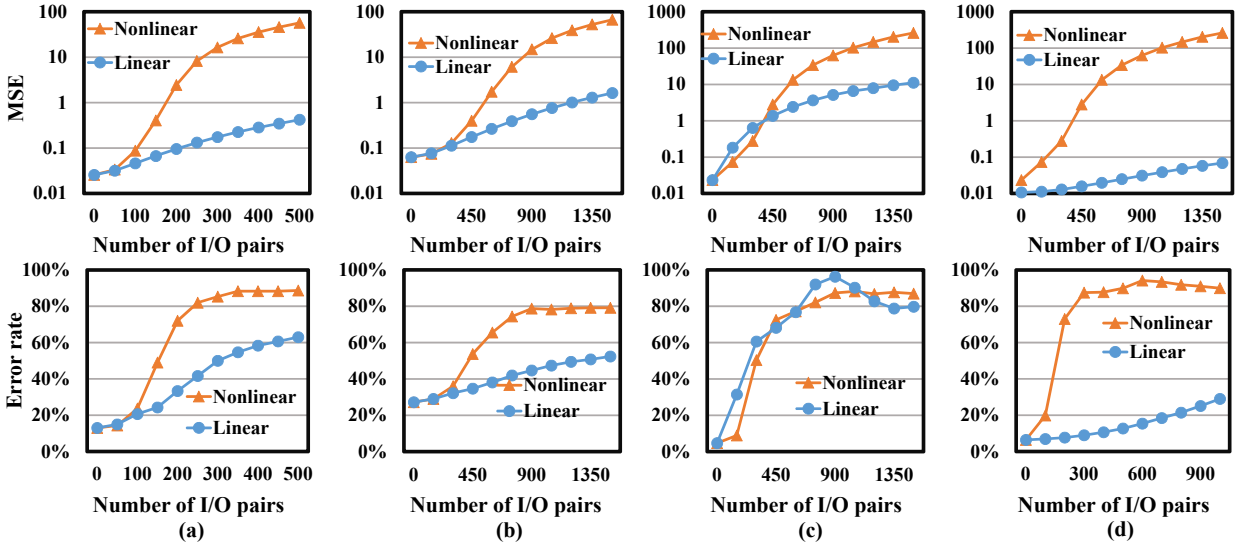


Fig. 8. Comparison between the naïve and the revised designs using MSE and error rate: (a) *Digit*, (b) *Faults*, (c) *Image*, and (d) *MNIST*.

comparisons between the naïve and the revised designs are summarized in Fig. 8, where x-axis represents the number of I/O pairs and y-axis shows the mean square error (MSE) and error rate, respectively, as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}|\boldsymbol{o}_i - \boldsymbol{t}_i|^2. \tag{20}$$

where $\boldsymbol{o}_i$ denotes the final output obtained from the last layer of the network.

The y-axis of the figure is in logarithmic scale for better view. MSE is the absolute difference between target results and classification results and hence, has a smooth monotonous curve. Error rate does not necessarily linearly depend on the system degradation, so the curve may contain many inflection points that are caused by uncertainty in real classification task.

As we can see from Fig. 8, the degradation curve of the revised design is much more nonlinear compared to the naïve design. The low error rate region at the beginning provides the usability for normal users, and the rapidly increasing portion guarantees the protection of the model. Take *Digit* as an example. As we can see from Fig. 8 (a), the error rate of the revised design keeps below 20% before obtaining 100 I/O pairs, and it increases to over 70% between 100 to 200 I/O pairs. At the same time, the error rate of the naïve design increases from below 20% to 60% gradually throughout the whole process without showing significant nonlinearity. For *Digit*, *Faults*, and *Image*, the average error rate increases within 50 I/O pairs are less than 5%. For *MNIST*, the error rate increases faster but still demonstrates the nonlinearity for both normal users and model protection. Furthermore, one can design the system to balance between usability and protection.

We also quantitatively analyze the nonlinearity using the correlation coefficient:

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2}\cdot\sqrt{\sum(y-\bar{y})^2}}. \tag{21}$$

Here x denotes the evaluating operations and y denotes evaluation index, i.e., error rate or MSE. $r \in (0,1)$ of which 1 represents the highest linearity, and 0 represents the highest nonlinearity. Then, we define the nonlinearity index as $1-r$. We take the curve from initial error rate to 60% of its maximum into consideration, because that is the part reflecting the change in first-order derivative. The result is shown in Table II. We can notice that the nonlinearity index of the revised design is much higher than that of the naïve design. The average increase in degradation rate for MSE is 179.93% and for error rate is 288.99%.

We also investigated the influence of network depth on the nonlinearity. We choose different neural networks with 1, 2, and 3 hidden layers to run on all the benchmarks. The result of *Digit* is shown in Fig. 9. The x- and y-axis are normalized to 0 to 1, respectively, for better observation of nonlinearity. The nonlinearity indexes are 0.809, 0.897 and 0.971 for 1 layer, 2 layers and 3 layers, respectively. The result confirms with our discussion in Section IV-C: The nonlinearity in degradation increases with the depth of neural networks. Another similar observation is that if the number of neurons in each layer is larger, the error rate increase is faster. This is shown in Fig. 8 (d). Note that the dimension of the first weight matrix of *MNIST* is "784×64" , which is much larger than other datasets. The error rate increase of the revised design is significant compared with the naïve design, with almost the same summed absolute changes of weights. This means that revised design does not introduce extra calibration

TABLE II
NONLINEARITY INDEX OF MSE AND ERROR RATE, BETWEEN NAÏVE AND REVISED DESIGNS.

		Digit	*Faults*	*Image*	*MNIST*
MSE	Naïve	0.0320	0.0509	0.0294	0.0401
	Revised	0.1015	0.1303	0.1010	0.0814
Error rate	Naïve	0.0250	0.0122	0.0243	0.0322
	Revised	0.0923	0.0762	0.0954	0.0546

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/TCAD.2019.2937817, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

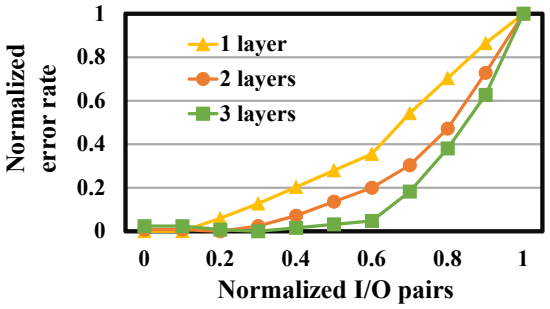


Fig. 9. Impact of network depth, varying from 1 hidden layer to 3 hidden layers.

cost, which is propotional to the summed absolute changes of weights. Detailed discussion can be found in Section V-F.

C. Accuracy of Replicated Model

In this section, we will mimic the replication attack and evaluate the effectiveness of our proposed design. In the simulation, we assume the best-case scenario for an attacker: 1) All the I/O pairs chosen by the attacker are the same as the one in the original training samples. 2) The model chosen by the attacker is the one with the best replication performance, e.g., SVM for *Digit*, Random Forest for *Faults*, similar to the analysis in Figure 2. Other simulated models include K-Nearest Neighbors and feedforward neural network. We show that even under the best-case scenario, an attacker cannot obtain the correct outputs due to the obsolescence effect of the system.

Fig. 10 summarizes our results where the x-axis is the I/O pairs and the y-axis is the accuracy. We compare the accuracies of the replication model when attacking three systems:

1) The original system without obsolescence property (*Original* in the figure). As the number of I/O pair increases, the accuracy should increase monotonously.
2) The system with naïve design (*Linear* in the figure). The accuracy should not increase monotonously. Ideally, it will increase to a peak and then decrease.
3) The system with revised design (*Nonlinear* in the figure). The accuracy should decrease earlier.

In the initial phase, all three systems achieve a similar trend of accuracy increase, because this period belongs to the low error rate region as we can observe from the curve in Fig. 8. The accuracies of the naïve and revised systems then both drop, while the accuracy of the original system increases. The accuracy of the naïve system drops more slowly compared with the revised system. We also observe that the highest accuracy of the revised system is always lower than that of the naïve system. For example, in Fig. 10 (a), the theoretical maximum accuracy (the attackers will hardly achieve this since they have no idea when to stop training) of the revised design is 78.5% while the one of the naïve design is 85.4%. Considering the fast degradation rate of the accuracy after reaching the maximum, the proposed revised design is the most resilient one to replication attacks among all the three designs.

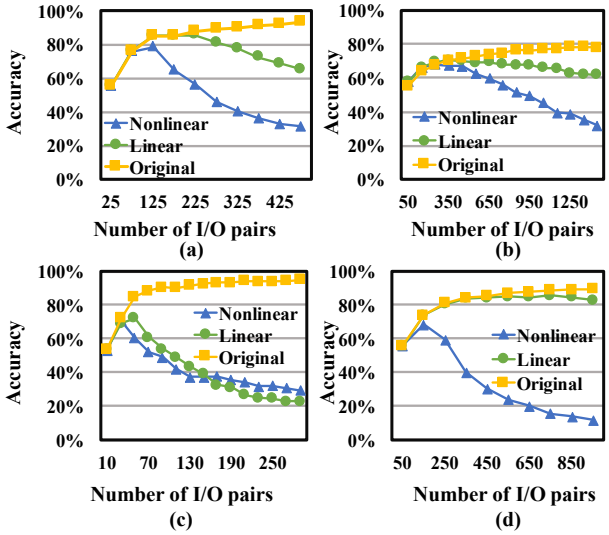


Fig. 10. Replication accuracy comparison between nave and revised designs: (a) *Digit*, (b) *Faults*, (c) *Image*, and (d) *MNIST*.

D. Ranges of Memristor Resistance

In the real design of MNCS, the resistance range of memristors greatly impacts on the performance of the system, including the accuracy drop incurred by the limitation of the precision, the change of the accuracy degradation rate due to different memristor obsoleting speeds, and the change of power consumption. The resistance levels of memristors can be tens of kiloohms to hundreds of kiloohms based on different models and techniques [31]. A higher resistance levels leads to a lower power consumption since $p = \frac{v^2}{r}$, where v is normally a constant for a chip, e.g., $1.2V$. However, the low working current incurred by the high resistance which is vulnerable to the noise current generated from sneak path effect.

In this section, we include experiments with different memristor resistance ranges to show the impact of memristor resistance on the robustness, effectiveness, and power consumption of our proposed design. Experiments are performed on *MNIST* with a fully-connected MLP (two hidden layers with 64 neurons for each in order to be consistent with the previous configuration) on our proposed revised design.

1) Resistance range analysis: To better demonstrate the impacts introduced by resistance range, we carefully tune the parameters to approximate the characteristics of real devices. We select the relatively flat portion of the derivative of conductance as shown in Fig. 3 (b) by applying a threshold of $t_{th} = 0.5$. The time ranges from 0 to $t_{th} \times t_{max}$. We choose

TABLE III
COMPARISON OF POWER CONSUMPTION CONTRIBUTED BY MEMRISTOR.

settings	Power (W)	I/O pairs
1k/100k	0.71	1
1k/300k	0.70	523
1k/500k	0.90	326
1k/700k	1.13	231
0.5k/150k	1.39	513
2k/600k	0.35	527
3k/900k	0.23	585

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/TCAD.2019.2937817, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

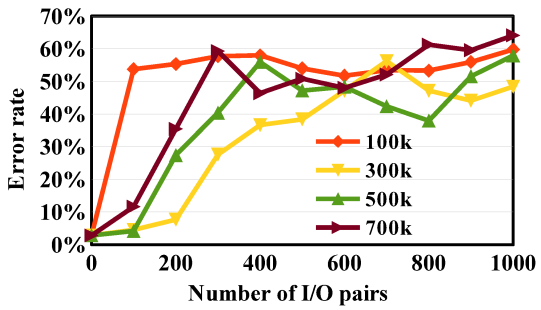


Fig. 11. System degradation comparison with different memristor resistance ranges, $R_L = 1k\Omega$.

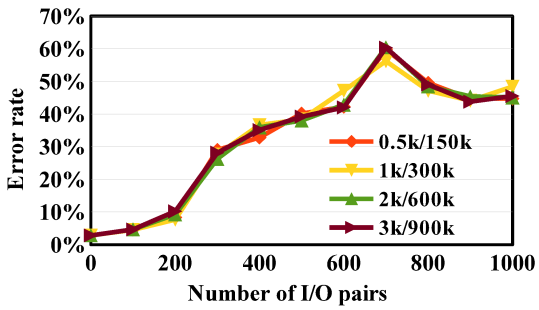


Fig. 12. System degradation comparison with different memristor resistance ranges, $\frac{R_H}{R_L} = 300$.

to use $R_L = 1k, R_H = 300k$ as our baseline. The parameters, e.g., obsoleting speed, mapping scalar, are also chosen in the baseline.

We first experiment on the accuracy degradation of the MNCS with the increase of I/O pairs under different R_H values, which is inverse proportional to the minimum allowable network weights (minimum and maximum weights are mapped to R_H and R_L, respectively). Fig. 11 compares the accuracy degradation curves of the MNCS when the R_H values ranges from $100k\Omega$ to $700k\Omega$. Here $R_L = 1k\Omega$ is a constant value. When $R_H = 100k\Omega$, the system does not function correctly even when the number of I/O pairs is small, experiencing a high error rate. This is due to the nature that the original weights of the model is majorly distributed around zero (neural networks tend be very sparse because of the regularization [32]). When small weights are forcely mapped to relative large conductances (e.g., when $R_H = 100k\Omega$), it will break the model's function. For other values, as R_H becomes larger, the conductance curve goes steeper; The memristors become obsoleted at a higher speed and the system degrades faster.

We also experimented on different (R_L, R_H) pairs with the same ratio. The ratio between the maximum and minimum weights represents the range of the weight distribution. Fig. 12 shows the system accuracy degradation w.r.t. the number of I/O pairs when the ratio $\frac{R_H}{R_L} = 300$. The result shows that the accuracy degradation speed and nonlinearity varies very little when the resistance range changes as long as the ratio stays at the same level.

2) Power consumption analysis: The energy consumed by a MNCS before its accuracy degrades to a certain level also changes with different memristor resistance ranges, as shown in Table III. Here we simulate the energy consumption of the MNCS when its error rate increases 40%. The corresponding numbers of I/O pairs that are needed to achieve such degradation are also listed in the table.

The first 4 rows of the table correspond to the configurations from Fig. 11 and the last 3 rows (including row 2) correspond to the configurations from Fig. 12. Under each setting, we analyze the power of the system when the error rate exceeds 40%, while all the wordlines are applied with $1V$ voltage. When the resistance ratio is fixed, the power consumption of the MNCS decreases as the memristor resistance range increases. This is consistent with our previous discussion. When the R_L is fixed, the power consumption of the MNCS increases when the R_H increases. This is because the system degrades faster with a larger resistance range. Therefore the overall conductance is larger when the error rate exceeds 40%, indicating a higher power consumption.

E. Robustness Analysis: Effect of Process Variation

In this section, we will evaluate the impact of memristor process variation including the device surface area and thickness [33]. We assume the memristor conductance follows a Gaussian distribution in general. For illustration purpose, we only include the results of *digit* in Fig. 13.

Process variation incurs the increase in the error rate of the system. Take the revised design as an example. The initial error rate when the number of I/O pairs is zero increase from 13% ($\sigma = 0$) to 16% and 18% when the standard deviations $\sigma = 1$ and 2, respectively. However, all the error rates approach to the same level when the number of I/O paris increases. We also calculated the nonlinearity index for each case, which are included in Fig. 13 too. In our revised design, the nonlinearity increases when the process variation increases. In naïve design, however, the nonlinearity always keeps at a low level when the process variation varies.

The results show that our revised design is robust to process variations and indeed its nonlinearity benefits from such variability. The initial accuracy degradation of the system can be partially compensated by enhanced training techniques such as variation-aware training [34].

F. Calibration Overhead

The cost of weight calibration operations discussed in Section IV-B is mainly determined by $\sum_{i,j} |w_{ij} - w'_{ij}|$, which is the summed discrepancy between the changed conductance and the fully trained conductance of the memristors.

Each operation cycle of our proposed secured MNCS consists of two periods – evaluating operation and system calibration. To calculate the calibration time overhead, we quantitatively measure the percentage of the calibration time T_{cal} over the whole operation cycle $T_{eval} + T_{cal}$ as:

$$OH_{cal} = \frac{T_{cal}}{T_{eval} + T_{cal}}. \tag{22}$$

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/TCAD.2019.2937817, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

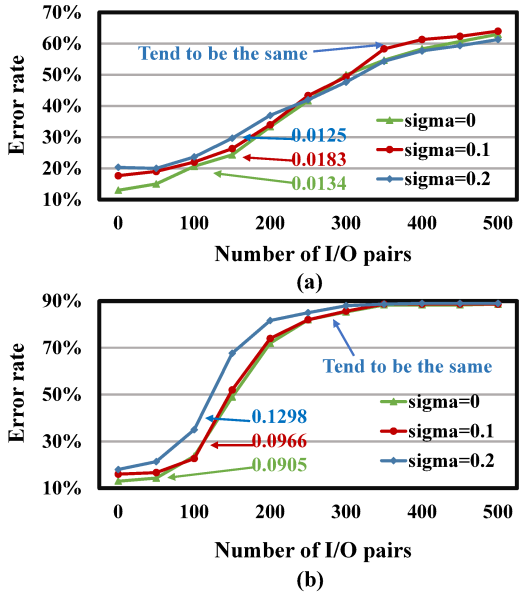


Fig. 13. System degradation comparison with different memristor process variation: (a) Naïve design and (b) Revised design.

Here the evaluation time T_{eval} and the calibration time T_{cal} can be further expressed as:

$$T_{eval} = \frac{N_{eval}}{f_{eval}} = \frac{g(\epsilon)}{f_{eval}}, \tag{23}$$

$$T_{cal} = \sum t_{cal}|w_{ij} - wij'|. \tag{24}$$

Here ϵ is error rate threshold. The system needs calibration when the error rate approaches ϵ. $g(\epsilon)$ is the number of test operations that can be performed before the system error rate reaching ϵ. f_{eval} is the frequency of evaluating operations, t_{cal} is the unit calibration time, and $t_{cal}|w_{ij} - w'_{ij}|$ denotes the time to calibrate w_{ij} from w'_{ij}.

For example, in the revised design of Fig. 8(a), if we set ϵ to 20%, when the error rate increases from original 13% to ϵ, only 100 I/O pairs can be applied. The overhead comparison between the naïve and revised designs can be found in Table IV. Here T denotes the difference between the initial error rate and the error rate threshold. In the above case, $T = 7\%$. Our result shows that the revised design incurs a very marginal increase in the calibration cost compared with the naïve design. This is because the conductance changing speed is almost the same for the both designs as indicated by Eq. (13) and Eq. (14).

G. Analysis on Attacks with Prior Knowledge of Defense Mechanism

As discussed in Section IV-D, the attacker may be aware of our defense mechanism in advance. In this section, we perform experiments based on this assumption (new attack model) using the revised design and *MNIST*.

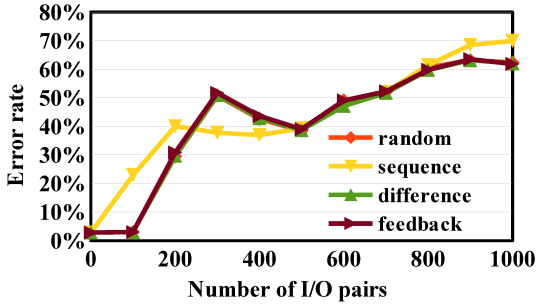


Fig. 14. System degradation comparison, applying different defending-aware attacks.

1) Manipulation of the Inputs: Previously, we applied random voltage pulses to a memristor crossbar once there is an input sample sent in in order to obtain a smooth and controllable accuracy degradation curve. However, the auxiliary random number generator could be a potential security issue and also introduce extra overhead and delay. In this section, we remove the randomness incurred by random voltage pulses, which means the speed of memristor shift is directly linked to input samples (voltage pulses). Thus, the system satisfies the requirement of the new attack model.

In Fig. 14 we compared the effectiveness of different attack techniques, including 1) sending inputs in random order (*random*), 2) sending inputs from zeros to nines (*sequence*), 3) sending inputs with largest difference (*difference*) and 4) sending inputs using feedback from results (*feedback*). The result indicates that *sequence* leads to the fastest system degradation, while the other three techniques have similar results among which *difference* has slightly slower system degradation. This phenomenon can be explained as follows: *sequence* sends similar inputs together to the system and hence cause fast accumulation of weight changes in the same direction. *distance* and *feedback* both try to alleviate such accumulation by sending samples with large differences between each other and impacting the weight changes differently. Nonetheless, for all the four attack techniques, the error rate of the MNCS quickly raises after 100 I/O pairs and effectively prevent the system from being attacked by learning the model.

2) Attack With Prior Knowledge: Without losing generality, we assume the attacker can get the output from the softmax layer as:

TABLE IV
COMPARISON OF CALIBRATION OVERHEAD (%).

B \ T		5%	10%	20%
Digit	Naïve	28.05	28.14	28.30
	Revised	29.15	29.09	29.24
Faults	Naïve	9.28	9.28	9.40
	Revised	9.22	9.24	9.26
Image	Naïve	50.65	50.36	50.54
	Revised	50.53	50.49	50.66
MNIST	Naïve	18.66	18.77	18.94
	Revised	19.67	19.67	19.67

$$P(y=j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \text{ for } j = 1, 2, ..., K. \quad (25)$$

In the digit classification task, $K = 10$ since there are 10 classes. $P(y = j)$ is the probability (also known as confidence score) that the system classify the input as class j. When we calculate the accuracy, we take the class with the highest probability and compare it with the label t.

$$accuracy = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(\boldsymbol{y}_i == \boldsymbol{t}_i). \quad (26)$$

where n is the size of test set. In *MNIST*, for example, $n = 10000$.

The attacker could analyze the output combinations and flip the labels of the I/O pairs they get, expecting to gain a better performance of the replicated model. Even though the attacker knows the defense mechanism, his/her choices are very limited.

First, the attacker can randomly flip the output label. However, the chance that the wrong label is flipped to the correct one is very low since the accuracy is usually very high. Assume the original model has an accuracy of 90% and the attacker wants to experiment on 100 samples. In this case, the attacker may want to flip 10 wrong labels out of 100 samples. Since there are C_{100}^{10} possible combinations of flips, even the possibility that these 10 flips cover 5 wrong label is extremely low (the chance is $C_{10}^{5} \times C_{90}^{5} / C_{100}^{10} = 0.06\%$). The fact that the original model is being compromised exacerbates the possibility. The above analysis demonstrates that the accuracy is more likely to get worse (correct labels are changed to the wrong label) when the attacker flips the label.

Second, it might be reasonable to flip the labels with lower confidence scores since these labels tend to be wrong. However, this statement is not true because the original model is being compromised, thus lowering the confidence even for correct labels. Besides, it is not feasible to put the weight-changing function of the original model into the replicated model's loss function since there is no connection between the original model and the replicated model.

In conclusion, the attacker can only manipulate the inputs to the original model in order to alleviate the system degradation, in which case we already know the proposed defense mechanism will still work.

VI. Related Works

Security attacks against learning models can be mainly categorized into two types: exploratory attack (exploitation of the classifier) and causative attack (manipulation of training samples) [35]. Causative attack denotes the situation when attackers are able to manipulate the training set and therefore change the parameters of the target model. In contrast, exploratory attack does not change the parameters of the target model. Replication attack also belongs to exploratory attack since the attack occurs in the inference phase, thus not modifying the model's parameters. Replication attack is also known as "model stealing attack" in some literature. Tramér *et al.* demonstrated simple and efficient attacks that can extract target ML models for popular model classes including logistic regression, neural networks, and decision trees [36]. Later, Juuti *et al.* proposed a new technique to detect model extraction which analyses the distribution of successive queries from a client and identifies deviations from a Gaussian distribution [37]. However, rather than detection, our method can physically thwart replication attack without introducing much overhead from both software and hardware sides. There are also some works on physical degradation-based defense which refers to hardware measures that enforce physical usage bounds through intentional degradation of the hardware. Rahmati *et al.* used the SRAM decay phenomenon to measure time for batteryless embedded devices in order to throttle response rates to adversarial accesses [38]. Deng *et al.* provided statistical guarantees on system-level usage based on the probabilistic wearout models. [39]. Some other hardware approaches can restrict data accesses through self-destructing circuits [40], [41].

VII. Conclusion

In this paper, we propose a design to prevent memristor-based neuromorphic computing system (MNCS) from being attacked by replicating the function of the model. We propose an approach across device-, circuit-, and architectural-levels to thwart this attack. We study the influences of process variation, different activation functions and cost functions, and the resistance ranges of memristors towards the effectiveness of our proposed approaches. We also estimate the power consumption and the calibration overhead under different settings. Compared to the naïve design, our revised design has a higher nonlinearity index, e.g., 179.93% on MSE and 288.99% on the error rate, indicating a more effective defense. Further analysis proves that our method can protect the MNCS even when the attacker knows our defense mechanism in advance.

Acknowledgment

The presented works were supported by NSF CCF-1615475 and AFRL FA8750-18-2-0057.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[3] S. Kumar, "Introducing qualcomm zeroth processors: Brain-inspired computing," https://www.qualcomm.com/news/onq/2013/10/10/introducing-qualcomm-zeroth-processors-brain-inspired-computing, 2013.

[4] M. Hu, H. Li, Q. Wu, and G. S. Rose, "Hardware realization of bsb recall function using memristor crossbar arrays," *IEEE/ACM Design Automation Conference*, pp. 498–503, 2012.

[5] B. Lendon, "Iran says it built copy of captured u.s. drone," http://www.cnn.com/2014/05/12/world/meast/iran-u-s-drone-copy/, 2014.

[6] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[7] T. K. Ho, "Random decision forests," *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 1, pp. 278–282, 1995.

[8] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[9] A. Mazady, M. T. Rahman, D. Forte, and M. Anwar, "Memristor pufa security primitive: Theory and experiment," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 2, pp. 222–229, 2015.

[10] J. Rajendran, G. Rose, R. Karri, and M. Potkonjak, "Nano-PPUF: A Memristor-Based Security Primitive," *in the Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 84–87, 2012.

[11] G. Khedkar, C. Donahue, and D. Kudithipudi, "Towards leakage resiliency: memristor-based aes design for differential power attack mitigation," in *SPIE Sensing Technology+ Applications*. International Society for Optics and Photonics, 2014, pp. 911 907–911 907.

[12] S. Yu, Y. Wu, and H.-S. P. Wong, "Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory," *Applied Physics Letters*, vol. 98, no. 10, p. 103514, 2011.

[13] T. Chang, S.-H. Jo, and W. Lu, "Short-term memory to long-term memory transition in a nanoscale memristor," *ACS nano*, vol. 5, no. 9, pp. 7669–7676, 2011.

[14] S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla *et al.*, "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013, pp. 1–10.

[15] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, "An efficient and scalable semiconductor architecture for parallel automata processing," *Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3088–3098, 2014.

[16] M. Hu, H. Li, Y. Chen, Q. Wu, and G. S. Rose, "Bsb training scheme implementation on memristor-based circuit," *IEEE Computational Intelligence for Security and Defense Applications*, pp. 80–87, 2013.

[17] B. Liu, Y. Chen, B. Wysocki, and T. Huang, "The circuit realization of a neuromorphic computing system with memristor-based synapse design," in *Neural Information Processing*. Springer, 2012, pp. 357–365.

[18] S. Kannan, N. Karimi, O. Sinanoglu, and R. Karri, "Security vulnerabilities of emerging nonvolatile main memories and countermeasures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 1, pp. 2–15, 2015.

[19] C. Helfmeier, D. Nedospasov, C. Tarnovsky, J. S. Krissler, C. Boit, and J.-P. Seifert, "Breaking and entering through the silicon," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 733–744.

[20] B. Liu, C. Wu, H. Li, Y. Chen, Q. Wu, M. Barnell, and Q. Qiu, "Cloning your mind: security challenges in cognitive system designs and their solutions," *IEEE/ACM Design Automation Conference*, pp. 1–5, 2015.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] D. Kuzum, S. Yu, and H. P. Wong, "Synaptic electronics: materials, devices and applications," *Nanotechnology*, vol. 24, no. 38, p. 382001, 2013.

[23] M. Hu, H. Li, and R. E. Pino, "Fast statistical model of tio_2 thin-film memristor and design implication," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 345–352, 2011.

[24] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[25] M. Elshamy, H. Mostafa, Y. H. Ghallab, and M. S. Said, "A novel nondestructive read/write circuit for memristor-based memory arrays," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 11, pp. 2648–2656, 2014.

[26] M. N. Bojnordi and E. Ipek, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 1–13.

[27] J. Zha, H. Huang, T. Huang, J. Cao, A. Alsaedi, and F. E. Alsaadi, "A general memristor model and its applications in programmable analog circuits," *Neurocomputing*, vol. 267, pp. 134–140, 2017.

[28] B. Li, Y. Wang, Y. Chen, H. H. Li, and H. Yang, "Ice: inline calibration for memristor crossbar-based computing engine," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–4.

[29] A. Asuncion and D. Newman, "Uci machine learning repository," 2007.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[31] A. Ascoli, F. Corinto, V. Senger, and R. Tetzlaff, "Memristor model comparison," *IEEE Circuits and Systems Magazine*, vol. 13, no. 2, pp. 89–105, 2013.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[33] D. Niu, Y. Chen, C. Xu, and Y. Xie, "Impact of process variations on emerging memristor," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. IEEE, 2010, pp. 877–882.

[34] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: variation-aware training for memristor x-bar," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 15.

[35] M. Barreno, B. Nelson, A. D. Joseph, and J. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[36] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 601–618.

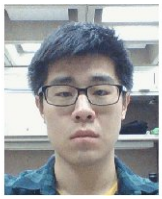
[37] M. Juuti, S. Szyller, A. Dmitrenko, S. Marchal, and N. Asokan, "Prada: protecting against dnn model stealing attacks," *arXiv preprint arXiv:1805.02628*, 2018.

[38] A. Rahmati, M. Salajegheh, D. Holcomb, J. Sorber, W. P. Burleson, and K. Fu, "Tardis: Time and remanence decay in sram to implement secure protocols on embedded devices without clocks," in *Proceedings of the 21st USENIX conference on Security symposium*. USENIX Association, 2012, pp. 36–36.

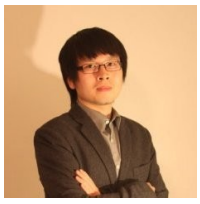
[39] Z. Deng, A. Feldman, S. A. Kurtz, and F. T. Chong, "Lemonade from lemons: Harnessing device wearout to create limited-use security architectures," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 361–374, 2017.

[40] N. Banerjee, Y. Xie, M. M. Rahman, H. Kim, and C. Mastrangelo, "From chips to dust: The mems shatter secure chip," in *2014 IEEE 27th International Conference on Micro Electro Mechanical Systems (MEMS)*. IEEE, 2014, pp. 1123–1126.

[41] J.-W. Han, M.-L. Seol, Y.-K. Choi, and M. Meyyappan, "Self-destructible fin flip-flop actuated channel transistor," *IEEE Electron Device Letters*, vol. 37, no. 2, pp. 130–133, 2015.



Chaofei Yang received B.S. from Tsinghua University and M.S. from University of Pittsburgh in 2017. He is a Ph.D. student in the Electrical and Computer Engineering department at Duke University. His research interests include Deep Learning Security and Neuromorphic Computing.



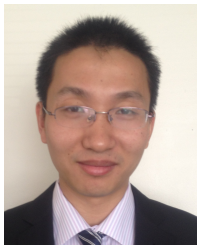
Beiye Liu is currently an applied scientist in Amazon. He received his B.S. from Southeast University, Nanjing, China. In 2016, Beiye graduated from University of Pittsburgh, Electrical and Computer Engineering department with Ph.D. Degree. His research interests include neuromorphic computing system, deep learning algorithms, deep learning hardware optimization, and nature language processing. His work has been published in conferences including ICCAD, DAC, ICCD, and ISCAS. Beiye has also served as review committee members of Embedded systems letters, ISQEA, Neurocomputing etc. He received Richard Newton Young Student Fellow in 2013 and best paper nomination in 2015 DAC.

Hai (Helen) Li (M'08-SM'16-F'19) received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, and the Ph.D. degree from the Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA, in 2004. Dr. Li currently is the Clare Boothe Luce Associate Professor with the Department of Electrical and Computer Engineering at Duke University, Durham, NC, USA. Prior to it, she was with Qualcomm Inc., San Diego, CA, USA, Intel Corporation, Santa Clara, CA, Seagate Technology, Bloomington, MN, USA, the Polytechnic Institute of New York University, Brooklyn, NY, USA, and the University of Pittsburgh, Pittsburgh, PA, USA. She has authored or co-authored more than 200 technical papers in peer-reviewed journals and conferences and a book entitled Nonvolatile Memory Design: Magnetic, Resistive, and Phase Changing (CRC Press, 2011). Her current research interests include neuromorphic architecture for brain-inspired computing systems, machine learning and deep neural network, memory design and architecture, and architecture/circuit/device cross-layer optimization for low power and high performance. Dr. Li is currently a Distinguished Lecturer of the IEEE CAS society and a distinguished speaker of ACM. She is a distinguished member of the ACM. Dr. Li is a recipient of the NSF Career Award, DARPA Young Faculty Award (YFA), and TUM-IAS Hans Fisher Fellowship from Germany. She received seven best paper awards and additional seven best paper nominations from international conferences. Dr. Li serves as Associate Editor of IEEE TCAD, IEEE TVLSI, IEEE TCAS-II, IEEE TMSCS, ACM TECS, IEEE CEM, ACM TODAES, and IET-CPS. She was the General Chair or Technical Program Chair of multiple IEEE/ACM conferences and the Technical Program Committee members of over 30 international conference series.

Yiran Chen (M'04-SM'16-F'18) received B.S and M.S. from Tsinghua University and Ph.D. from Purdue University in 2005. After five years in industry, he joined University of Pittsburgh in 2010 as Assistant Professor and then promoted to Associate Professor with tenure in 2014, held Bicentennial Alumni Faculty Fellow. He now is a tenured Associate Professor of the Department of Electrical and Computer Engineering at Duke University and serving as the director of NSF IndustryUniversity Cooperative Research Center (IUCRC) for Alternative Sustainable and Intelligent Computing (ASIC) and co-director of Duke Center for Evolutionary Intelligence (CEI), focusing on the research of new memory and storage systems, machine learning and neuromorphic computing, and mobile computing systems. Dr. Chen has published one book and more than 350 technical publications and has been granted 93 US patents. He serves or served the associate editor of several IEEE and ACM transactions/journals and served on the technical and organization committees of more than 50 international conferences. He received 6 best paper awards and 12 best paper nominations from international conferences. He is the recipient of NSF CAREER award and ACM SIGDA outstanding new faculty award. He is the Fellow of IEEE and Distinguished Member of ACM, a distinguished lecturer of IEEE CEDA, and the recipient of the Humboldt Research Fellowship for Experienced Researchers.

Mark Barnell (M'09) received the B.S. degree of Optical Engineering from University of Rochester in 1987 and the M.S. degree in Computer Science from SNUY Polytechnic Institute in 2000. He is a Senior Computer Scientist with the U.S. Air Force Research Laboratory, high performance computing systems branch (AFRL/RITB). Mr. Barnell currently is the HPC Director for AFRLs Information Directorate Computing and Communications Affiliate Resource Center (ARC) and Agile High Performance Systems (AHPS) Program Manager. His areas of expertise include high performance computers, embedded computing, persistent wide area surveillance, distributed and next generation architectures.

Qing Wu (M'01) received the B.S. and M.S. degrees from the Department of Information Science and Electronic Engineering at Zhejiang University, Hangzhou, China, in 1993 and 1995, respectively, and the Ph.D. degree from the Department of Electrical Engineering at the University of Southern California in 2002. Currently, he is a Senior Electronics Engineer at the United States Air Force Research Laboratory (AFRL), Information Directorate (RI). Before joining AFRL, he was an Assistant Professor in the Department of Electrical and Computer Engineering at State University of New York, Binghamton. His research interests include large-scale neuromorphic computing circuits and systems, highperformance computing architectures, energy-efficient embedded computing.



Wujie Wen (M'16) is currently an assistant professor in the department of Electrical and Computer Engineering at Florida International University (FIU), Miami, FL. He received his Ph.D. from University of Pittsburgh in 2015. He earned his B.S. and M.S. degrees in electronic engineering from Beijing Jiaotong University and Tsinghua University, Beijing, China, in 2006 and 2010, respectively. Before he joined FIU, he also worked with AMD and Broadcom for various engineer and intern positions. His current research interests include deep learning hardware acceleration/security, neuromorphic computing, and circuit-architecture design for emerging memory technologies. His works have been published in top-tier conferences (e.g. HPCA, DAC, ICCAD, DATE, ICPP, HOST, ECCV, AAAI, CVPR etc). Dr. Wen is the associate editor of Neurocomputing and serves as the General Chair of ISVLSI 2019 (Miami), Program Chair of ISVLSI 2018 (Hong Kong), as well as program committee for many conferences such as DAC, ICCAD, ASP-DAC etc. He received best paper nominations from ASP-DAC2018, ICCAD2018, DATE2016 and DAC2014. He was also the recipient of the 49th DAC A. Richard Newton Graduate Scholarship, the most prestigious Ph.D. scholarship (one awardee per year) in EDA society and 2015 DAC Ph.D. forum best poster presentation. His research is sponsored by NSF, AFRL and Florida Center for Cybersecurity etc.

Jeyavijayan (JV) Rajendran (S'09-M'15) is an Assistant Professor in the Department of Electrical and Computer Engineering at the Texas A&M University. Previously, he was an Assistant Professor at UT Dallas between 2015 and 2017. He obtained his Ph.D. degree in the Electrical and Computer Engineering Department at New York University in August 2015. His research interests include hardware security and emerging technologies. His research has won the NSF CAREER Award'17, the ACM SIGDA Outstanding Ph.D. Dissertation Award'17, three Student Paper Awards (ACM CCS 2013, IEEE DFTS 2013, and IEEE VLSI Design 2012). He organizes the annual Embedded Security Challenge, a redteam/blueteam hardware security competition and has cofounded Hack@DAC, a student security competition colocated with DAC. He is a member of IEEE and ACM.