SCA: A Secure CNN Accelerator for Both Training and Inference

Lei Zhao
University of Pittsburgh
Pittsburgh, PA, USA
lez21@pitt.edu

Youtao Zhang
University of Pittsburgh
Pittsburgh, PA, USA
zhangyt@cs.pitt.edu

Jun Yang
University of Pittsburgh
Pittsburgh, PA, USA
juy9@pitt.edu

***Abstract*—Convolutional neural networks (CNNs), while being widely deployed to edge devices, face increasingly requirements for IP protection, i.e., the protection of the models and their weights. This becomes particularly challenging for those that demand post-deployment training to enhance inference performance. Existing schemes focus mainly on IP protection at the inference phase, and lack the ability to extend to the training phase. In this paper, we propose SCA, a secure CNN accelerator that exploits stochastic computing to achieve IP protection at both training and inference phases. We propose hybrid stochastic addition and weight remapping to further optimize space utilization and design robustness. Our experimental results show that SCA effectively prevents pirating the CNN IP from the authorized devices. In addition, it achieves 4.8× and 34.2× speedups and 84.3% and 98.5% energy reductions over a non-secure baseline and an inference-only secure baseline, respectively.**

***Index Terms*—security, convolutional neural networks, accelerator, stochastic computing**

I. Introduction

Convolutional neural networks (CNNs) are widely deployed to edge devices for inference tasks in computer vision [1], speech recognition [2] and many other domains. There are increasingly requirements for IP (intellectual property) protection on CNNs [3]–[5], i.e., we need to protect the models and their weights. For example, an object recognition system deployed in auto-driving cars may carry significant financial interests. Recent works proposed to integrate device dependent information as fingerprint in the CNN models, which prevents them from functioning on pirated devices [4], [5]). Other protection schemes include watermarking and encryption based schemes. The former can only verify if a CNN model is pirated from a particular provider while the latter tends to introduce significant decryption latency and energy consumption overheads.

Many CNN models demand post-deployment training, i.e., there is a need to train the deployed model to achieve improved inference performance. It is usually not preferable to send the edge devices to the manufacturer for further training There are two reasons. (i) Post-deployment training may take advantage of end users' private data, which creates privacy concerns if sending such data back to the device manufacturer. (ii) It is often physically difficult to retrieve the devices and send to the manufacturer. Unfortunately, existing IP protection schemes focus mainly on the inference phase and lack the ability to extend to the training phase. For example, we need to know the DRAM error mask in AEP [4] to train a device-dependent model. While the error mask is known to the server, it is not visible to the end devices, which prevents training the model effectively on the edge devices. To summarize, it is a major challenge to achieve post-deployment IP protection for end devices that demand both training and inference.

In this paper, we propose SCA, a secure CNN accelerator that exploits stochastic computing to achieve IP protection at both training and inference phases. To the best of the author's knowledge, SCA is the first work that uses hardware fingerprints for model protection in both training and inference phases. We summarize our contributions as follows.

- We propose SCA to exploit the precision difference between stochastic format and binary format representations as device dependent fingerprints, and integrate the fingerprints in the CNN models such that the trained model functions well only on the target devices.
- We propose to optimize the baseline SCA with weight remapping and hybrid stochastic addition. Weight remapping periodically uses different offsets in stochastic bit stream conversion such that it effectively prevents the model from learning unnecessary data representation features to improve the robustness of security in training phase. Hybrid stochastic addition explores the precision loss property of `MUX` based additions to reduce memory access and computational overheads to meet the tight budget requirements in edge computing.
- We evaluate the proposed SCA scheme. Our experimental results show that SCA effectively prevents pirating the CNN IP from the authorized devices. In addition, it achieves 4.8× and 34.2× speedups and 84.3% and 98.5% energy reductions over a non-secure baseline and an inference-only secure baseline, respectively.

II. Background

A. Convolutional Neural Network

A Convolutional Neural Network (CNN) is a stacked structure of layers. The major computation in a CNN layer is weighted sum of the product between input and model weights: $b_j = \sum_i a_i \cdot w_{i,j}$. The weights ($w_{i,j}$) are learned from the training phase and the inputs (a_i) are from the previous layer.

This work is supported by NSF CCF-1910413, NSF CCF-1718080, NSF CCF-1617071 and NSF CCF-1725657.

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

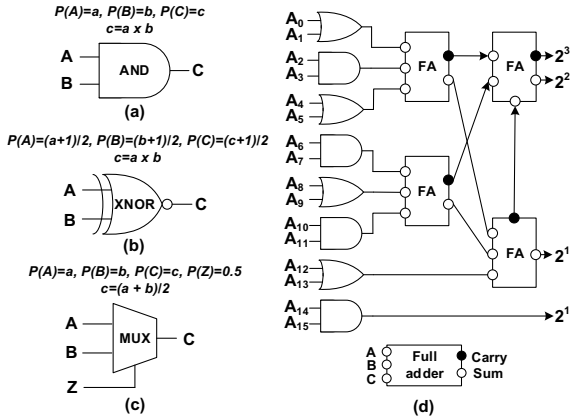

Fig. 1: Stochastic Computing Implementation.

CNN Training is an iterative process in which each iteration consists of a *forward phase* and a *backward phase*. The forward phase feeds input images sequentially through layers. The output of previous layer is the input to the next. The final layer computes a loss ($Loss$) between its output and the ground truth of the input. The backward phase propagates the loss layer by layer. The gradients of each layer's weights are also computed ($\frac{\partial Loss}{\partial W}$) during the backward phase. At the end of backward phase, all the weights are updated $W = W - \eta\frac{\partial Loss}{\partial W}$, where η is the learning rate.

B. Stochastic Computing

Stochastic Computing (SC) represents a number by using a random bit steam, in which the probability of the appearance of 1s indicates the represented value. For example, the bit stream $X = 011001001$ represents 0.4 because the probability $P(X = 1) = 0.4$. Since values are represented by probabilities, the representation for a value is not unique. A multiplication between two stochastic numbers can be implemented using bit-wise `AND` operations, as shown in Figure 1(a). However, only values in the range $[0, 1]$ can be encoded using the above method (called unipolar format). The bipolar format can represent values in the range $[-1, 1]$ by scaling it in $[0, 1]$. For example, the value $x = -0.4 \in [-1, 1]$ is first scaled to $y = (x + 1)/2 = 0.3 \in [0, 1]$, then encoded into $P(Y = 1) = 00101010$. The multiplications for bipolar stochastic numbers are implemented using $XNOR$ gates (Figure 1(b)). For both unipolar and bipolar formats, additions are commonly calculated by multiplexers (`MUX`s), as shown in Figure 1(c). A third bit stream Z is required. $P(Z = 1)$ is set to a constant 0.5 to give the same probability to select a bit from either A or B. So, the result is a scaled version of addition: $P(C) = P(Z)P(A)+(1-P(Z))P(B) = \frac{1}{2}(P(A)+P(B))$. However, since half of the information in the input bit streams are lost, `MUX` based addition suffers accuracy reduction. Another way to perform additions more accurately is to use Approximate Parallel Counter (APC). APC counts the number of 1s in input bit streams with some errors to trade off logic gate counts. Figure 1(c) shows an example APC converting a 16-bit stochastic number to 4-bit binary number. Note that the output of APC is in binary format.

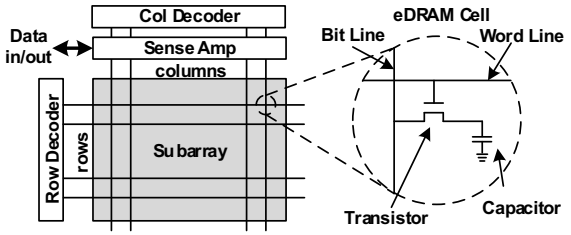

Fig. 2: eDRAM organization.

C. eDRAM and eDRAM PUFs

Figure 2 illustrates the eDRAM structure. eDRAM cells are organized as rectangular subarrays, consisting of rows and columns. Each cell is composed of a capacitor and an access transistor. A cell stores one bit data according to the amount of charge in the capacitor. eDRAM cells suffer from capacitor's charge leakage over time and thus demand periodical refreshes.

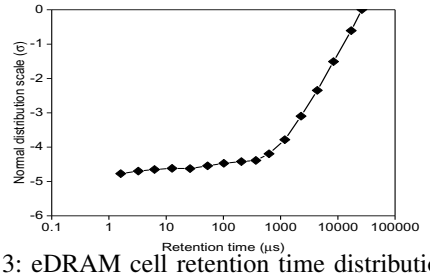

Fig. 3: eDRAM cell retention time distribution [6].

Due to process variations, different eDRAM cells leak charge at different speeds. Recent studies showed that the bit error rates increase with reduced refresh frequency [6], as shown in Figure 3. Because weak cells leak charge much faster and the distribution of these weak cells are device dependent, many prior works [5], [7] have exploited it to device physical unclonable functions (PUFs) that map a set of input parameters to unique, device-specific signatures that can be generated repeatably and reliably.

III. Design

A. Overview

A high level overview of the workflow is shown in Figure 4(a). It consists of two phases — *design phase* and *working phase*. (i) In the *design phase*, machine learning experts train the CNN model with the best efforts on the server side, i.e. using binary format weights and inputs. The trained weights are in cleartext, which are vulnerable to attacks. We then deploy the model to the target device. Given the target device, we first reduce its refresh frequency to generate approximately 50% retention failures in some selected rows of the on-chip eDRAM buffers. We record the used refresh frequency and row addresses as the device's fingerprint as store this information in the SC conversion (SCC) unit shown in 4(b). We then restore the default refresh frequency in all subsequent operations. We next embed the fingerprint in the CNN model such that the device (i.e., the accelerator) is ready to be deployed. (ii) In the *working phase*, the user exploits the accelerator to accomplish its inference task. In addition,

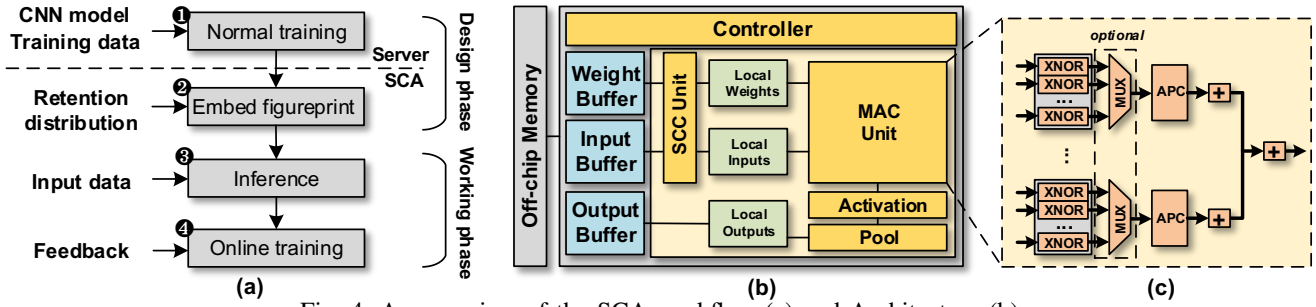

Fig. 4: An overview of the SCA workflow (a) and Architecture (b).

the user may collect new inputs in the field and the desired inference results. Such data can be used to train the accelerator for improved inference accuracy.

A high level overview of the SCA architecture is shown in Figure 4(b). The fingerprint-embedded weights and the input data are stored in binary format in off-chip memory before execution. For inference and training, SCA first loads the weights and the input into on-chip eDRAM buffers. To exploit stochastic computing (SC), the SC conversion (SCC) unit converts the buffered binary weights into stochastic format using the previously selected eDRAM rows, and store them in the *Local Weights buffer*. The MAC (multiply-accumulate) unit performs stochastic multiply-accumulation operations, shown in Figure 4(c). Finally, the results are written back to off-chip memory through Local Output and on-chip eDRAM output buffer. Whenever the weights and intermediate results between layers are in eDRAM buffers or off-chip memory, they are in binary format. In this way, the CNN model is kept safe during the whole process.

Intuitively, since the fingerprint is embedded in the binary weights, conducting CNN inference using the binary weights on an error-free device leads to poor inference accuracy. The converted stochastic format weights carry device dependent fingerprint and thus produce accurate inference results only if the selected eDRAM rows match those on the target device.

B. SC based Protection

Given device dependent SC conversion plays a key role in protecting the CNN IP, we next elaborate its implementation details.

1) Generating Device Dependent Bit Streams: Traditional SC generators (converting binary numbers into stochastic bit streams) use random number generators or linear feedback shift registers to generate one bit per cycle. Given an N-bit binary number needs to be converted to a 2^N-bit stream to achieve sufficient precision, existing SC circuit designs tend to incur large area and performance overheads, e.g., the SC generator takes as much as 90% of the total area in [13].

SC uses probability to represent values, i.e., it is the number of 1s rather than their positions that determine the overall computation accuracy. SC is intrinsically error tolerance as having one more or fewer bit incurs little impact. Consequently, instead of using RNGs or LFSRs, SCC leverages the PUF capability of eDRAM rows to generate device dependent random bit streams. As shown in Figure 5, when system starts up, we first fully charge all the cells in some selected eDRAM rows (referred to as victim rows in the rest of this paper) (❶). Then we drastically reduce refresh frequency to make approximately half of the cells flip the stored value through capacitor leakage (❷). The reduced refresh frequency can be determined according to Figure 3. During normal CNN computing, SCA still uses the default refresh frequency (❸). The startup phase (❶-❸) only executes once. A victim row can be used multiple times to generate different SC numbers.

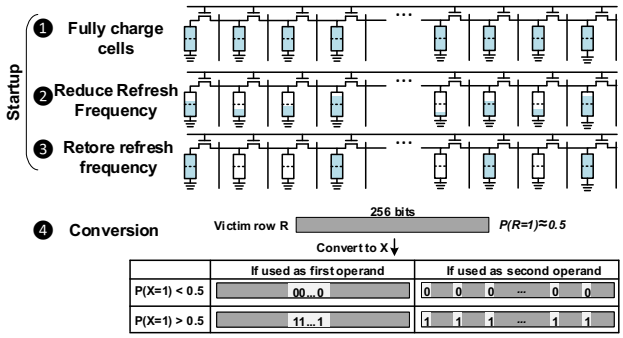

Fig. 5: Stochastic computing conversion.

When converting a binary number to stochastic format, a portion of the bit stream is filled with 0s or 1s depending on the converted value. As shown in Figure 5 ❹, the victim row R is generated through steps ❶ to ❸, thus $P(R=1) \approx 0.5$. If the probability of 1s in the target bit stream X is less than 0.5, 0s are filled into specific positions of the bit stream. Otherwise 1s are filled. The bits that are filled by 0s or 1s are similar to [11]. That is, if the generated bit stream is used as the first operand (e.g. weights), we fill consecutive 0s or 1s in the middle of the bit stream. If the generated bit stream is used as the second operand (e.g. input), 0s or 1s are scattered along the bit stream. The number of bits filled are determined by a look-up table.

2) Embedding Device Dependent Information in NN models: CNNs are intrinsically error-tolerant algorithms because they are typically over-parameterized. We modify the conventional training method to leverage this characteristic to embed the device dependent information into CNN weights. In each iteration of ❷ in Figure 4(a), the binary format weights W_b are first converted to stochastic format weights W_{sc}, which then participate in the forward phase to calculate the loss. In backward phase, gradients are calculated with respect to the binary format weights, i.e. $\frac{\partial Loss}{\partial W_b}$, as well as the update,

i.e. $W_b = W_b - \eta \frac{\partial Loss}{\partial W_b}$. The basic principle is that CNN training only finds local minima on the loss surface, if a weight value is modified, other weights can change correspondingly to mitigate the impact and forces the training to find other local minima. This process is only performed for a few iterations (10 to 20) on the accelerator.

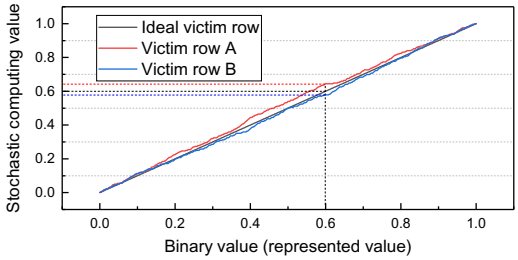

Fig. 6: Stochastic computing values using different victim rows.

3) Model Protection: Because of process variation, relatively weak cells in the victim rows tend to leak charge faster during startup phase (as shown in Figure 5). As a result, the distribution of 1s along each row is device dependent. As a consequence, the converted stochastic value is also device dependent and usually not exactly the same as the binary value. Figure 6 shows the values converted from two different randomly selected victim rows. The X and Y axes are the binary value and converted stochastic value, respectively. We can see that different victim rows have different precision to represent the same value. Figure 6 also shows an example of converting 0.6 from binary format to stochastic format using these two victim rows. Assume the authorized device has victim row A. The actual weight value that is used in the CNN computing is 0.64. However, if an adversary extracts the value from either off-chip memory or on-chip eDRAM buffers, he/she can only get the binary value 0.6, which degrades the CNN prediction accuracy. Even if the adversary also use the same SC conversion method but on another device, because it is very unlikely that any of its victim row has the same error distribution as victim row A, the converted value may be 0.57 assuming victim row B is the selected row in the pirate device. Table I compares the inference accuracy of these three cases.

TABLE I: Model Pretection

	On Same Device	No SC Computation	On Another Device
MNIST-MLPS	99.73%	57.98%	60.25%

Previous works ([4], [5]), once loaded on-chip, the weights are no longer under any protections since fingerprint information is removed by memory errors. So the weights can not be swapped out to off-chip memory if get updated in training. In SCA, whenever being swapped off-chip, the weights are converted back to binary format by APC, thus protected by the precision difference between stochastic and binary format value.

C. Robust Protection

Because CNNs are usually over-parameterized, in addition to learning the classification task of the target problem the model may also have the capability to learn the difference patterns between stochastic and binary format values. As a result, the binary format weight values (W_b) tends to be closer to the converted stochastic format weight values (W_{sc}) when sufficiently enough training iterations are applied. The lines in Figure 7 show the accuracy and 2-norm distance between values of these two formats with the training iteration proceeds on MNIST-MLPS. Eventually, the capability of model protection introduced in Section III-B vanishes, as shown by the red line in Figure 7.

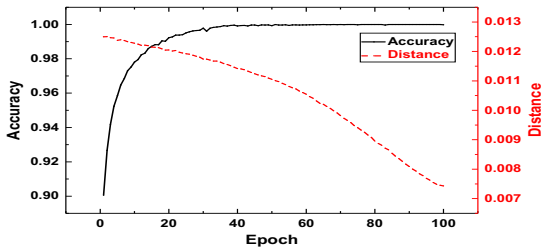

Fig. 7: Accuracy and distance between binary format weights and SC format weights on MNIST-MLPS.

To prevent the vanishing of differences between stochastic and binary values, we propose a remapping technique. For each weight conversion, instead of inserting into the middle of the bit stream, we use an offset to insert the consecutive 0s or 1s. The offset changes periodically overtime. However, a sudden change on all the weight conversion offset will drop the model accuracy. Therefore, we change a subset of weight conversion offset each time. More specifically, in each epoch we change one layer's weight conversion offset at a time. For example, in epoch one, we change the offset for the first layer's weight conversion while keep other layer's weight conversion offset unchanged. In epoch two, we only change the second layer's weight conversion offset, etc.

D. Space-Optimized Stochastic Computing

SC uses simple logic to perform computations, but it brings a new problem of exponentially increased data representation overhead which also aggravates the burden on memory bandwidth. For example, the weight matrix size of AlexNet, a state-of-the-art CNN, increases from 139MB in binary format to 238GB in stochastic format, which easily exceeds the capacity of on-chip memories in most CNN accelerators. In this section, we explore the optimizations on MAC operations to alleviate this problem.

previous CNN accelerator designs using SC [11], [12] prefer APC based adders to `MUX` based adders due to their higher precision. The low precision of `MUX` based adders are caused by the dropped information (e.g., the grey bits in Figure 8(a) are dropped).

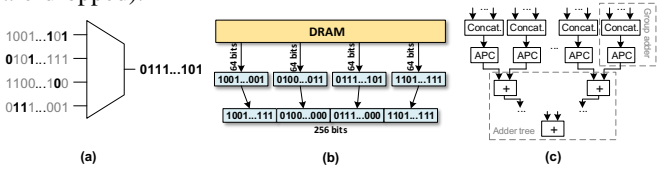

Fig. 8: Convention and proposed SC Addition.

We propose a hybrid addition implementation that partition the summation inputs into groups. Inside each group, we

first use MUX based adders to get the sum, which is then converted into binary format through an APC based adder. The binary format outputs of all groups then go through an adder tree to get the final sum. We tested different group sizes and found using group size of 4 can keep CNN's accuracy drop with 2%. Since the MUXs drop some bits, which is a waste of the previous computations (multiplications between weights and inputs in CNN). We can use shorter bit streams in multiplication, and use concatenation to substitute the MUX based adders. In the example shown in Figure 8(b), we use bit streams of 64 bits instead of 256 bits for group size of 4 to save memory bandwidth and computation. Figure 8(c) shows the entire MAC unit.

IV. Methodology

To evaluate SC's impact on CNN accuracy, we developed a custom CNN framework with C++ and CUDA and replaced all the computations in CUDA kernels to SC bit-wise operations. We tested on three widely adopted image classification datasets: MNIST [8], SVHN [9], and Cifar10 [10]. In addition to CNN models for each dataset, we also include three multilayer perceptrons of different sizes (MLP-S/M/L) in our benchmarks. The details of the network structures are listed in Table II.

TABLE II: Datasets and Networks

Benchmark	Neural Network
MNIST-MLPS	250-10
MNIST-MLPM	500-250-10
MNIST-MLPL	1000-500-250-10
MNIST-CNN	con5x20-pool2-500-10
SVHN-CNN	conv5x32-pool2-conv5x64-pool2-256-10
Cifar10-CNN	conv4x32-pool2-conv5x32-pool2-conv5x64-pool2-500-10

We used Design Compiler with 45nm FreePDK library to synthesize the SC logic to get latency, area and power parameters. SRAM and eDRAM parameters were extracted from CACTI. We developed an in-house simulator to evaluate SCA's performance and energy consumption.

We compare SCA with two baselines. SC-CNN [14] is a CNN accelerator implemented with stochastic computing with no security support. AEP [4] uses memory errors as hardware fingerprint to protect CNN weights, but no protection for training phase, and its computation is based on conventional binary arithmetic. We evaluate two schemes of SCA:

- SCA-1. This is our basic SCA implementation as elaborated in Section III-B.
- SCA-2. This implementation integrates the hybrid addition optimization.

We also investigate SCA's protection robustness in Section V-C;

V. Results

A. Characteristics

Table III lists the characteristics of a SCA chip using 45nm technology. A SCA chip can accomadate 2048 group adders. Each group adder can multiply 4 stochastic numbers and sum up to one binary output. The outputs of all the 2048 group adders go through the adder tree to form the final summation. SCA uses three eDRAMs as on-chip buffers for input, weights and output, respectively. The converted stochastic numbers are stored in three SRAM buffers for faster access.

TABLE III: SCA Characteristics

Units	Number/Size	Area	Power
Group Adder			
XNOR	256	$6553.6um^2$	3.53nW
APC	1	$3886.3um^2$	0.195uW
Total	1	$10439.9um^2$	0.195uW
MAC			
Group adder	2048	$21.38mm^2$	400.5uW
Adder tree	1	$1.71mm^2$	1.0328mW
Chip			
MAC	1	$23.1mm^2$	1.432mW
eDRAM Buffers	3×2MB	$17.6mm^2$	1.328W
SRAM Buffers	3×16KB	$0.204mm^2$	59.49mW
SC unit Total	1	$13.65mm^2$	233mW
Other	1	$3.97mm^2$	1.05W
Chip total	1	$59mm^2$	2.68W

B. Protection On Weights

Table IV shows SCA's protection effectiveness on inference. The CNN models are trained using the method described in Figure 4(a). The second column shows the inference accuracy on the same device as used for training (i.e. authorized devices). The third column shows the inference accuracy if the CNN models are running on pirate devices which cannot generate the same victim rows that are used in training. The last column shows the inference accuracy if the CNN models are used in conventional binary arithmetic based computing. All the accuracy numbers are normalized to that when training with binary format values.

From the last two columns of Table IV, we can see a significant drop on all the benchmarks. No matter the pirate devices use the same SC conversion method or use binary arithmetic directly, the tested benchmarks can not generate satisfiable inference accuracy, which renders the effectiveness of SCA's weight protection scheme. When the tested benchmarks run on the same devices as used in training, only a negligible inference accuracy loss (less than 1%) is observed (first column in Table IV). This proves the fidelity of SCA's protection scheme.

TABLE IV: Inference Accuracy

	Target Device	Pirate Device	No SC
MNIST-MLPS	99.73%	57.98%	60.25%
MNIST-MLPM	99.69%	21.57%	24.75%
MNIST-MLPL	99.99%	20.99%	26.38%
MNIST-CNN	99.07%	14.27%	81.82%
SVHN-CNN	99.84%	69.04%	60.9%
Cifar10-CNN	99.99%	41.86%	52.44%

C. Protection robustness in Training

Figure 9 shows the effectiveness of our remapping technique for training. The solid black line shows the test accuracy normalized to conventional binary arithmetic based CNN. The

dashed red line shows the distance between stochastic format weights and binary format weights. The first few epochs, where accuracy experiences an observable increase, shows when the models are learning the classification tasks. At first the distance is big due to the random initialization of weights. The distance drops with the training going on due to the excessive learning capability of the model to also learn the value differences between stochastic and binary formats. When the models reach the highest accurary (convergence), the distance stops decreasing thanks to the periodically changed remapping technique for SC conversion. Because all the weights will use a new conversion offset after some epochs, the model can not find a constant difference pattern between the stochastic and binary format values. Other benchmarks show similar trends as MNIST-CNN.

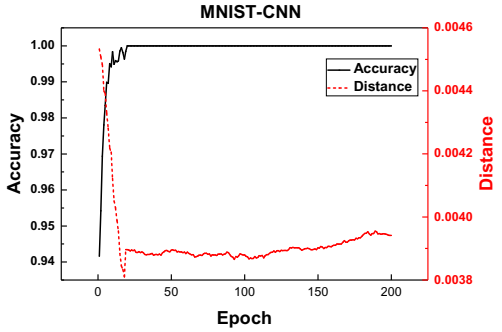

Fig. 9: Protection robustness in training.

D. Performance

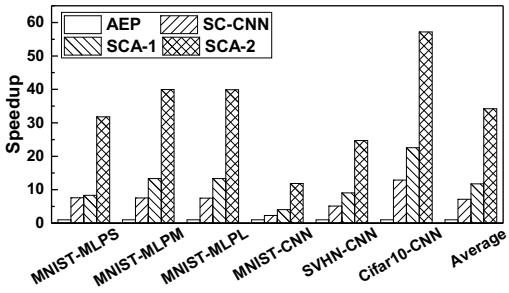

Fig. 10: Speedup.

Figure 10 shows the speedups normalized to the AEP baseline. AEP uses conventional binary arithmetic for computation. The speedup of SC-CNN mainly comes from the faster SC based multiplications. Although SC-CNN employs a counter to replace the RNG for one of the multiplication operands, the other operand still needs a RNG to make sure the bit stream is randomized. On contrary, SCA-1 generates SC numbers for both multiplication operands by simple memory read and table lookup. So a larger speedup can be observed for benchmarks that require more data conversion. For example, Cifar10 and MNIST-MLPL have more weights than other benchmarks, so they have higher speedups than other benchmarks. On averagae, SCA-1 achieves 11.7× speedup over AEP. SCA-2 uses shorter bit streams (64 bits instead of 256 bits), so both computation and memory access can be saved. Thus, SCA-2 can achieve 34.2× speedup on average. Compared to SC-CNN, SCA-2 achieves 4.8X speedup.

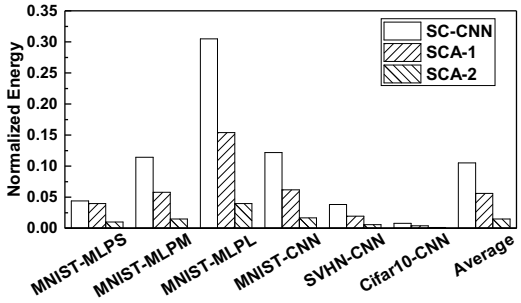

Fig. 11: Normalized energy consumption.

E. Energy

Figure 11 shows the results normalized to AEP. SC-CNN saves energy by replacing multipliers with simple gates. On average, SC-CNN's total energy consumption is 10.5% of AEP. SCA-1 integrates SC conversion into the memory read process. Because SC conversion is needed for both input and weights whenever they are read from off-chip memory, SCA-1 further reduces the energy consumption to 5.6% of AEP. SCA-2 not only eliminates MUXs in addition operations, but also reduces the memory bandwidth burden by using shorter bit streams, thus SCA-2 only consumes 1.4% energy of AEP and 15.7% energy of SC-CNN.

VI. Conclusion

In this paper, we proposed SCA, a SC based secure CNN accelerator. SCA uses eDRAM refresh errors as hardware fingerprint to generate bit streams for SC conversion. SCA provides protection for both training and inference phases. To prevent the stochastic format weights get closer to their binary format values, we proposed a remapping technique to insert 0s or 1s to different part of the bit stream for SC conversion. We also proposed a hybrid addition method to avoid memory footprint explosion caused by long stochastic bit streams.

References

[1] A. Krizhevsky, *et al.*, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[2] G. Hinton, *et al.*, "Deep neural networks for acoustic modeling in speech recognition," in *IEEE Signal processing magazine*, 2012.

[3] Y. Uchida, *et al.*, "Embedding watermarks into deep neural networks," in *ICMR*, 2017.

[4] L. Zhao, *et al.*, "AEP: An error-bearing neural network accelerator for energy efficiency and model protection," in *ICCAD*, 2017.

[5] W. Li, *et al.*, "P^{3}M: a PIM-based neural network model protection scheme for deep learning accelerator," in *ASPDAC*, 2019.

[6] A. Agrawal, *et al.*, "Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip eDRAM modules," in *HPCA*, 2014.

[7] T. Christensen, *et al.*, "Implementing physically unclonable function (PUF) utilizing EDRAM memory cell capacitance variation," in *US Patent 8,300,450 B2*, 2012.

[8] Y. LeCun, *et al.*, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998.

[9] Y. Netzer, *et al.*, "Reading digits in natural images with unsupervised feature learning," in *NIPS*, 2011.

[10] A. Krizhevsky, *et al.*, "Learning multiple layers of features from tiny images," in *Citeseer*, 2009.

[11] S. Li, *et al.*, "SCOPE: A Stochastic Computing Engine for DRAM-Based In-Situ Accelerator," in *MICRO*, 2018.

[12] A. Ren, *et al.*, "Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing," in *ASPLOS*, 2017.

[13] W. Qian, *et al.*, "An architecture for fault-tolerant computation with stochastic logic," in *IEEE transactions on computers*, 2010.

[14] H. Sim, *et al.*, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *DAC*, 2017.