SRA: A Secure ReRAM-Based DNN Accelerator

Lei Zhao
University of Pittsburgh
Pittsburgh, PA, USA
leizhao@cs.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

Deep Neural Network (DNN) accelerators are increasingly developed to pursue high efficiency in DNN computing. However, the IP protection of the DNNs deployed on such accelerators is an important topic that has been less addressed. Although there are previous works that targeted this problem for CMOS-based designs, there is still no solution for ReRAM-based accelerators which pose new security challenges due to their crossbar structure and non-volatility. ReRAM's non-volatility retains data even after the system is powered off, making the stored DNN model vulnerable to attacks by simply reading out the ReRAM content. Because the crossbar structure can only compute on plaintext data, encrypting the ReRAM content is no longer a feasible solution in this scenario.

In this paper, we propose SRA – a secure ReRAM-based DNN accelerator that stores DNN weights on crossbars in an encrypted format while still maintaining ReRAM's in-memory computing capability. The proposed encryption scheme also supports sharing bits among multiple weights, significantly reducing the storage overhead. In addition, SRA uses a novel high-bandwidth SC conversion scheme to protect each layer's intermediate results, which also contain sensitive information of the model. Our experimental results show that SRA can effectively prevent pirating the deployed DNN weights as well as the intermediate results with negligible accuracy loss, and achieves 1.14× performance speedup and 9% energy reduction compared to ISAAC – a non-secure ReRAM-based baseline.

KEYWORDS

security, neural networks, ReRAM, accelerator

1 INTRODUCTION

Deep neural networks (DNNs) keep achieving superior inference accuracy over other machine learning methods, leading to their pervasive and dominant use in many application domains. The increasing demand for efficient DNN processing has prompted many accelerator designs. Besides the designs based on conventional CMOS technology [1–3], existing works have also demonstrated the use of emerging non-volatile technologies, such as metal-oxide resistive random access memory (ReRAM) [4], spin-transfer torque magnetic RAM (STT-RAM) [5], and phase change memory (PCM) [6], to design process-in-memory (PIM) computing engines for DNNs. Among them, the ReRAM crossbar structure is the most widely studied approach. By performing the computation in memory, the massive data movement can be avoided. Also, the crossbar-based PIM paradigm provides opportunities for much higher computation parallelism.

However, how to protect the DNN model after its deployment on accelerators remains a less addressed problem, especially for ReRAM-based designs. Firstly, due to its non-volatility, ReRAM does not need a continuous power supply to retain data. This makes the accelerator susceptible to new security vulnerabilities, for example, accessibility to the stored model if a device gets stolen. Secondly, because ReRAM's crossbar structure can only compute on plaintext data, it is not feasible to store the encrypted model on the crossbars. Recently, there are previous works [7–9] that utilize hardware characteristics as fingerprints to protect DNNs. Unfortunately, they can only protect the DNN model stored off-chip, thus only applicable to CMOS-based accelerators. Once the model is loaded on-chip, the weights are automatically converted into plaintext format, making the entire model at risk if the accelerator is based on ReRAM technology.

In this paper, we propose a secure ReRAM-based DNN accelerator (SRA) that stores DNN weights in an encrypted format while maintaining ReRAM's PIM capability. The multiplication is performed in the Stochastic Computing (SC) format, and all weights and inputs are represented in the form of bit streams. SRA uses the 1T4T crossbar structure [10] so that each physical bit line is sliced into segments, which can be activated separately. An arbitrary set of segments from a pair of two adjacent physical bit lines can be merged into a logical bit line, which stores the bit stream of a weight. Therefore, the actual weight value is encrypted by keeping the merging pattern confidential. We also propose to encrypt multiple weights on the same pair of physical bit lines, reducing the storage overhead of long bit streams.

Besides the deployed model weights, the intermediate results produced between layers also contain sensitive information of the DNN model. For example, the intermediate results of fully connected layers and convolutonal layers are a linear function of the inputs, and knowing the inputs and outputs of a linear function makes stealing the model a trivial problem. With the increasing DNN model sizes in recent years, it is unavoidable to offload the intermediate results to off-chip storage given limited on-chip resources on accelerators. Although the intermediate results can be protected by encrypting them in off-chip storage and decrypting them before loading on-chip, due to the extremely high memory bandwidth demand in DNN accelerators, this approach introduces significant latency and energy consumption overheads. In SRA, we propose a novel SC conversion scheme that not only generates random bit streams with a high bandwidth but also provides protection on the intermediate results.

In summary, we make the following contributions in this paper:

- To the best of our knowledge, SRA is the *first* secure ReRAM-based accelerator design that could protect the deployed model weights while maintaining the PIM capability.
- SRA proposes a high-bandwidth random bits generator that could also protect the intermediate results at the same time.
- The proposed weight encryption scheme also supports sharing bits among multiple bit streams, significantly reducing SC's storage overhead.

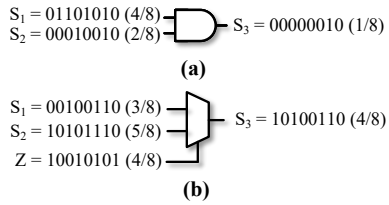


Figure 1: (a) SC multiplication using an AND gate. (b) SC addtion using a multiplexer.

- The experimental results show that in addition to effective protection, SRA also achieves performance improvement and energy consumption saving.

The remaining of this paper is organized as follows. Section 2 introduces the background of SC and the 1T4R ReRAM crossbar structure. Section 3 gives an overview of the SRA workflow. The detailed SRA design is described in Section 4. Section 5 elaborates our proposed SRA accelerator architecture. Experimental methodology and results are described in Section 6. Finally, the conclusion is made in Section 7.

2 BACKGROUND

2.1 Stochastic Computing

Stochastic Computing (SC) uses a bit stream to represent a number. The represented value is indicated by the ratio of 1s in the bit stream. For example, the bit stream 00010101 represents the value 0.375 (3/8) because 37.5% of the bits are 1. In SC format, multiplication is performed by an element-wise AND operation. Figure 1 (a) illustrates the multiplication of two bit streams S_1 and S_2, representing the values of 0.5 and 0.25, respectively. The ratios of 1 in S_1 and S_2 are denoted as $p_1 = 50\%$ and $p_2 = 25\%$. Because p_1 and p_2 are percentages, they can also be interpreted as the probability of observing a 1 at an arbitrary position in the bit streams. After the AND operation, the probability of observing a 1 for each position in the resulting bit stream is $p_1 \times p_2$. So S_3 represents the production of $S_1 \times S_2$. Adding S_1 and S_2 is implemented by a multiplexer and another random bit stream with 50% 1s, as shown in Figure 2 (b). This random bit stream Z controls the multiplexer to randomly select a bit from either S_1 or S_2. So the resulting bit stream contains the value of $\frac{1}{2}(p_1 + p_2)$, which is then multiplied by 2 to get the correct sum. However, because the multiplexer drops half of the information in S_1 and S_2, this addition usually incurs a large precision loss. So, many previous works [8, 11, 12] use more complex circuits (e.g., Approximate Parallel Counter) to convert S_1 or S_2 into the binary format first, and then use conventional adders to perform the addition. The major challenge that limits SC's practical applicability is its long bit length. For a n-bit binary number, SC needs 2^n bits to preserve the same precision. This means a 1-bit increase in precision requires an exponential increase in bit stream length.

2.2 1T4R ReRAM Crossbar Structure

A ReRAM device is a three-layer structure, consisting of a resistive switching layer sandwiched between the top and bottom electrodes, as shown in Figure 2(a). The cell's resistance can be programmed by *SET* and *RESET* operations to change the oxygen vacancy filament connecting the two electrodes. The oxygen vacancy filament determines the resistance between the two electrodes. A single ReRAM cell can store more than one bit by dividing the resistance range into multiple levels to achieve higher density. However, this requires more complex I/O circuits and poses a challenge to programming accuracy. Therefore, many works [13, 14] use single-bit cells in PIM designs for practical concerns.

ReRAM cells can be organized in a compact crossbar structure to achieve the smallest $4F^2$ cell size. Figure 2(b) shows the multiply-accumulate (MAC) operations taking place in a crossbar. The conductance (i.e., the reciprocal of the resistance) of the ReRAM cells are programmed according to the weight values. DNN's inputs are converted into voltages that are applied on the word lines. Thus, the current flowing out from each bit line conveys the MAC result between the inputs and weights. The single-bit-cell crossbar structure is an excellent match for SC multiplication if the columns in the crossbar and the inputs are all bit streams. The current flowing out from each bit line now conveys the count of 1s in the resulting bit stream, thus the product is automatically converted to binary format.

Because there are no access transistors to isolate adjacent cells, unactivated word lines or bit lines can induce leakage current (referred to *sneak current*) during access. Recently, [10] presented a 1T4R ReRAM crossbar structure to address the sneak current problem. In a 1T4R crossbar, each 4×4 sub-array is isolated from other cells in the crossbar by 4 dedicated transistors, which are fabricated underneath the sub-array. Figure 2 (c) and (d) illustrate the 2D and 3D layout, respectively. [10] fabricated a 1T4R test chip to show that the 4 transistors can be fully hidden underneath the area of the 4×4 ReRAM sub-array, thus still achieving the $4F^2$ cell size. Figure 2 (d) shows the schematic diagram of one 4×4 sub-array. Besides bit lines (BLs) and word lines (WLs), there are also two global select lines (GSLs) and two global word lines (GWLs). The GSLs are used to select which rows in the sub-array to access. The GWLs connect to other sub-arrays horizontally in the crossbar.

3 OVERVIEW

Both weights and inputs are computed in SC format in SRA. However, only positive numbers can be represented by the SC format introduced in Section 2.1 (called the *unipolar format*). Although there is also a *bipolar format* of SC representation that could encode both negative and positive numbers, the multiplication needs to be performed by XNOR operations which are not applicable to crossbar implementation. Fortunately, the ReLU activation in the DNNs filters out all negative values in each layer's outputs (i.e., inputs to the next layer), only weights may have negative values. In SRA, we only store the absolute values of the weights in the *unipolar format* SC bit streams while the sign bits are stored is a separate crossbar (called the sign bit crossbar in this paper). It is possible to expose the sign bits to the adversary, but only knowing the sign bits without the actual absolute weight values can not generate useful inference results.

Figure 3 shows the overview of SRA's workflow, which takes a pre-trained DNN, the training data, and a set of *GSL* vectors as inputs. Each *GSL* vector determines how the segments are merged into logical bit lines. The *Bit Stream Mapping* step (Section 4.2)

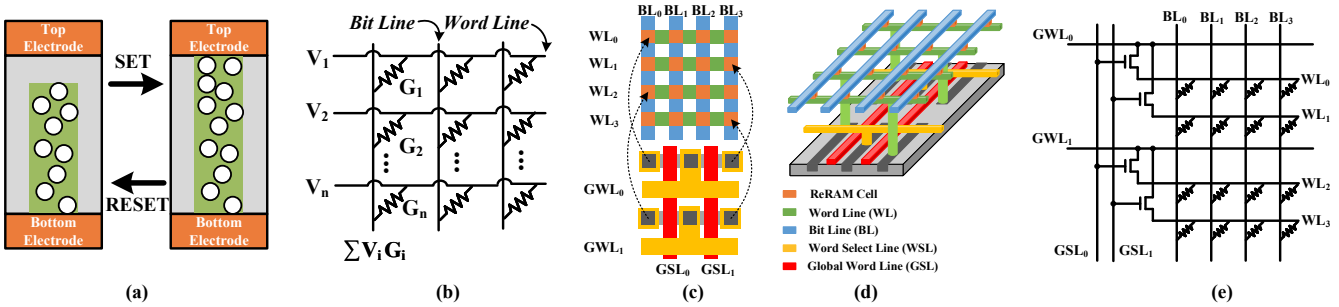


Figure 2: (a) ReRAM cell structure. (b) ReRAM crossbar structure. (c) 1T4R 2D layout. (d) 1T4R 3D layout. (e)1T4R schematic diagram.

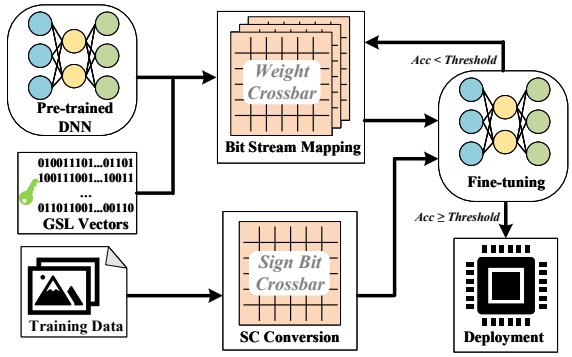


Figure 3: An overview of the SRA workflow.

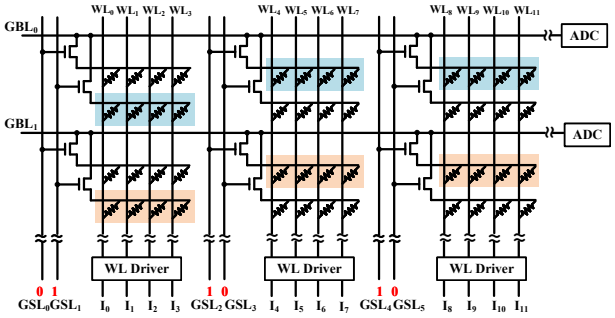


Figure 4: The rotated layout of the 1T4R structure.

uses a mixed-integer linear programming (MIP) model to map the weights onto the logical bit lines. Because the mapping introduces noises into the weights, the DNN model needs an iterative process of fine-tuning and mapping until the accuracy exceeds a predefined threshold. The *SC conversion* step (Section 4.3) converts each layer's inputs to SC format before computing with the weights. Note that all the *Bit Stream Mapping*, *SC conversion* and *Fine-tuning* steps are performed offline, and the ReRAM crossbars are only programmed once in the *Deployment* step.

4 DESIGN DETAILS

4.1 Encrypted Crossbar Design

In order to slice the bit lines into segments, we adopt the rotated layout of the original 1T4R structure to switch the functionality of the bit lines and word lines. Since the crossbar is a symmetrical structure and the ReRAM cells are acting as resistors during computation, the rotated layout does not need any changes inside the crossbars, only the positions of the word line drivers and the ADCs are exchanged. As shown in Figure 4, the ADCs are placed at the right of the crossbar while the word line drivers are at the bottom. In this new layout, we rename the original horizontal global word lines (*GWLs*) to global bit lines (*GBLs*), and the original bit lines (*BLs*) are called word lines (*WLs*).

In this new layout, each row in the crossbar is now divided into segments consisting of 4 cells. The *GSLs* control which segments are connected to the *GBLs* during computation. For the example shown in Figure 4, the first pair of *GSLs* selects the segments on the second and fourth rows to connect to the *GBLs*, the second pair of *GSLs* selects the segments on the first and third rows to connect to the *GBLs*, etc. As a result, all the blue cells in Figure 4 are merged to form a logical bit line to compute with the inputs I, and the results are accumulated on GBL_0. Similarly, all the orange cells in Figure 4 are merged to form another logical bit line to compute with the inputs I, and the results are accumulated on GBL_1. We restrict each pair of *GSLs* can only be 01 or 10, such that either the segments on the even rows or the segments on the odd rows will be connected to the *GBLs*. Therefore, only one bit in each *GSL* pair needs to be stored. In SRA, we use a bit vector to record the first bit in each *GSL* pair (i.e., GSL_0, GSL_2, ...). This bit vector is referred to as the *GSL vector* in this paper. The uncolored cells in the figure can form another two logical bit lines when all the *GSLs* negate their inputs. By keeping the *GSL* vector secret, the actual content of the logical bit lines is stored in the crossbar in an encrypted form.

4.2 Bit Stream Mapping

It has been shown in previous studies that DNNs can be quantized to 8 bits without losing accuracy. Thus, SRA uses 256-bit streams in SC format for weights and inputs to preserve the same precision of 8-bit binary format. For a 256×256 crossbar and a *GSL* vector, there are 256 logical bit lines, each consisting of 256 bits. It is straightforward to map 256 weights onto the crossbar by storing one weight on each logical bit line. However, this incurs a 32× storage overhead compared to the original 8-bit binary format. In this section, we propose a novel bit stream mapping strategy to store more than 256 weights on one crossbar to reduce the storage overhead.

We use multiple *GSL* vectors. Each *GSL* vector maps a different set of 256 weights on the crossbar. For n *GSL* vectors, a crossbar can store $256n$ weights, with every pair of adjacent physical bit lines constructing $2n$ logical bit line to $2n$ weights. Figure 5 shows an example of mapping 4 weights on the first two physical bit lines in the crossbar assuming $n = 2$. Each square in the figure represents a segment. S_i indicates the number of 1s in each segment. Since

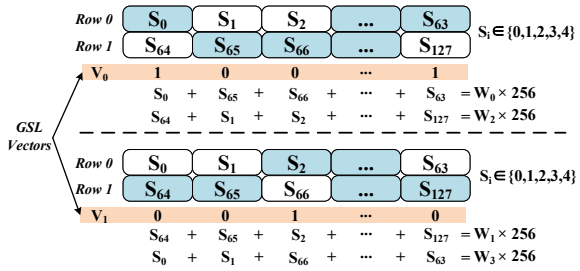


Figure 5: Mapping 4 weights on two rows in the crossbar.

a segment has 4 cells, S_i can only be 0, 1, 2, 3, or 4. V_0 and V_1 are the two *GSL* vectors. The first weight W_0 is mapped to the logical bit line determined by V_0, i.e, S_0, S_{65}, S_{66}, ..., S_{63}. The second weight W_1 is mapped to the logical bit line determined by V_1, i.e, S_{64}, S_{65}, S_2, ..., S_{127}. By using the negation of V_0 and V_1, we can get another two logical bit lines using the remaining segments to store W_2 and W_3. The left-hand side of the equations in Figure 5 represents the total number of 1s in each logical bit line. In order to correctly represent W_i, the total number of 1s in the logical bit lines needs to be $W_i \times 256$, which is the right-hand side of the equations. The mapping process is to determine the S_is to satisfy those equations. However, when n increases, it becomes hard to make all the equations to be true. We formulate this problem as a mixed-integer linear programming (MIP) model that minimizes the difference between the left-hand sides and the right-hand sides. In general, the MIP model for mapping $2n$ weights on the two adjacent physical bit lines can be written as follows:

$$\textit{Minimize} \quad \sum_{i=0}^{2n} D_i$$

Subject to:

$$S_j \in \{0, 1, 2, 3, 4\} \qquad (0 \le j \le 127)$$

$$S_j \ge 0 \qquad (0 \le j \le 127)$$

$$S_j \le 4 \qquad (0 \le j \le 127)$$

$$D_i \ge \left(\sum_{j=0}^{63} V_{i,j} \cdot S_j + \sum_{j=0}^{63} \overline{V_{i,j}} \cdot S_{j+64}\right) - W_i \times 256 \qquad (0 \le i \le n-1)$$

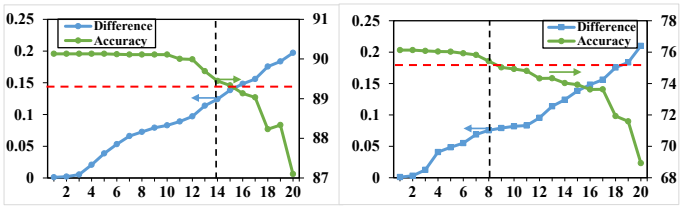
$$D_i \ge W_i \times 256 - \left(\sum_{j=0}^{63} V_{i,j} \cdot S_j + \sum_{j=0}^{63} \overline{V_{i,j}} \cdot S_{j+64}\right) \qquad (0 \le i \le n-1)$$

$$D_i \ge \left(\sum_{j=0}^{63} \overline{V_{i,j}} \cdot S_j + \sum_{j=0}^{63} V_{i,j} \cdot S_{j+64}\right) - W_i \times 256 \qquad (n \le i \le 2n-1)$$

$$D_i \ge W_i \times 256 - \left(\sum_{j=0}^{63} \overline{V_{i,j}} \cdot S_j + \sum_{j=0}^{63} V_{i,j} \cdot S_{j+64}\right) \qquad (n \le i \le 2n-1)$$

D_i represents the difference between the two sides of the equations. The first three constraints restrict the number of 1s in each segment to be integers between 0 and 4. Since linear programming can not use absolute values as constraints, two inequalities are used to represent the absolute difference. The next two constraints indicate the difference when mapping weights on logical bit lines determined by V_is. The last two constraints indicate the difference when mapping weights on logical bit lines determined by $\overline{V_i}$s.

When n becomes larger, it is harder to minimize the target. Figure 6 shows the mappings of Cifar10 and ResNet50 (model details are listed in Table 2) when n increases. The blue lines illustrate the average difference between the left-hand sides and right-hand sides after optimization, while the green lines show the inference accuracy. From the figure, we can see that different models have different tolerance on n. If we set the accuracy loss budget to 1%



(a) Cifar10 (b) ResNet50

Figure 6: Tolerance of n for Cifar10 and ResNet50.

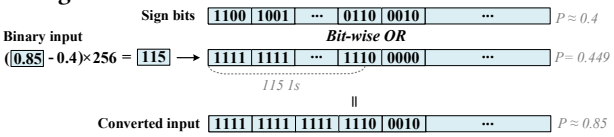


Figure 7: An example of converting 0.85 from binary format to SC format.

(marked by the red dashed line), n can be set to 14 in Cifar10 while n can not exceed 8 for ResNet which is also the worst case in all our tested DNN models. It is possible to use larger ns for different DNNs to achieve more storage savings, however, we conservatively set n to 8 for all DNN models for simplicity in evaluation.

4.3 SC Conversion

Conventional SC conversion uses linear feedback shift registers (LFSRs) to generate random bit streams with a specific percentage of 1s. However, LFSR-based SC conversion can only generate 1 bit at a time, introducing a long latency for each conversion. For faster SC conversion, We propose to reuse the sign bits of the weights as the source to generate random bit streams.

For each computation between the inputs and the weights, the weights' sign bits also need to be read out from the sign bit crossbar. Since the distribution of 1s in the sign bits of the weights is known beforehand, we can inject 0s or 1s into the sign bits to adjust the ratio of 1s according to the target input value. Figure 7 shows an example of converting the input 0.85 to SC format, assuming the ratio of 1s in the sign bits are 40% (represent a value of 0.4). In the example, 115 1s are injected into the sign bits consecutively starting from the first bit position. There may be other inject patterns by changing the start bit position of the consecutive 1s.

In order to protect the intermediate results, we use a hash table to select a different inject pattern for different inputs. Because 1s are not evenly distributed along the sign bits, using different inject patterns will introduce some random noises into the converted SC values. The hash table is kept confidential in an SRAM buffer on-chip and stored encrypted off-chip. Therefore, the adversary can only observe the binary format intermediate results stored off-chip while the SC format values participated in the computation are dynamically generated on-chip.

5 ARCHITECTURE

Figure 8 shows the architecture of the proposed SRA accelerator, which is attached to the system through the PCIe bus and works as a slave to process DNN tasks received from the CPU. The SRA accelerator is a tiled structure that uses a concentrated mesh to provide communications between the tiles. The details of one tile are shown on the right of Figure 8. Each tile has an eDRAM buffer

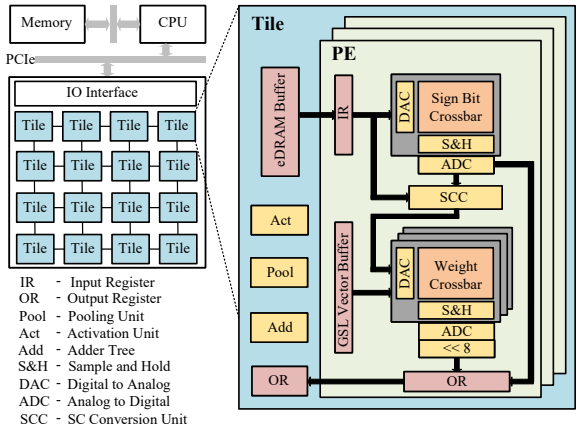


Figure 8: SRA accelerator architecture.

to store the binary format inputs of the DNN layer that is currently doing multiplications with the weights in the Processing Engines (PEs). An Output Register (OR) collects the PEs' multiplication results, which are then summed up in the adder tree. Each tile also has a Pooling unit for pooling layers and an Activation unit that implements the ReLU activation function. Each PE has a set of ReRAM crossbars. One of the crossbars stores the sign bits of the weights. The remaining crossbars are used for storing the SC format weights and performing multiplications with the inputs. Each PE also has an SRAM buffer to hold the *GSL* vectors. SRA uses 8 *GSL* vectors, each *GSL* vector has 64 bits. So, the size of the *GSL* vector buffer of each PE is 64B.

6 EXPERIMENTS

6.1 Methodology

We compare SRA with ISAAC[15] as it is the most widely used baseline in other ReRAM-based DNN accelerator designs. Comparing with ISAAC makes it easy to scale SRA's performance and energy consumption numbers to compare with other works. Table 1 lists the detailed parameters of an SRA chip. Most of the configurations are kept the same with ISAAC, except that we use 8-bit quantized pre-trained weights, 256×256 crossbars, and single-bit ReRAM cells in SRA. To get a fair comparison, we make the same changes to ISAAC as the baseline during evaluation. We follow the equation in [18] to scale the ADC power consumption for a different ADC resolution. The power parameters of SRAM registers and eDRAM buffers are obtained by simulation using CACTI[16] at 32nm process assumed in ISAAC. We use NVSim[17] to get parameters of ReRAM crossbars. We evaluate SRA's performance and energy consumption by using a custom cycle-accurate simulator to simulate DNN's layer-by-layer execution on SRA. The simulated accelerator runs at 1.2GHz.

We test SRA on four datasets: MNIST[19], SVHN[20], Cifar10[21] and ImageNet[22]. For SVHN and Cifar10, we construct custom neural network structures, whose details are listed in Table 2. We use LeNet-5[23] for MNIST. We test ImageNet on four networks – ResNet50[24], ResNeXt50[25], GoogLeNet[26] and DenseNet[27]. All these four networks are popular ones in the machine learning community. We use PyTorch for fine-tuning and accuracy evaluation. we use the PuLP library to solve the MIP optimizations.

Table 1: SRA accelerator parameters.

Unit	Spec	Power
PE		
ADC	num: 8; resolution: 8bits	16mW
DAC	num: 8×256; resolution: 1bit	8mW
S+H	num: 8×256	20uW
Xbar array	num: 8; size: 256×256; cell bits: 1	2.7mW
IR	size: 2KB	1.24mW
OR	size: 256B	0.23mW
GSL buffer	size: 64B	0.112mW
Shifters	size: 256	10uW
SCC	num: 1	0.02mW
PE Total	num: 1	28.4mW
Tile		
PE	num: 12	339.9mW
eDRAM	size: 64KB; banks: 2; width: 256	20.7mW
Bus	wires: 384	7mW
Router	flit size: 32; ports: 8	42mW
Activation	num: 2	0.52mW
Add	num: 1	0.05mW
Maxpool	num: 1	0.4mW
OR	size: 3KB	1.68mW
Tile Total	num: 1	412.3mW
Chip		
Tile	num: 168	69.2W
Hyper Tr	links: 4; freq: 1.6GHz	10.4W
Chip Total	num: 1	79.7W

Table 2: Benchmarks.

Dataset	Network
MNIST	LeNet-5
SVHN	conv3x32-conv3x32-pool-conv3x64-conv3x64-pool-conv3x128-conv3x128-pool-1024-512-10
Cifar10	conv3x128-conv3x128-conv3x128-pool-conv3x256-conv3x256-conv3x256-pool-conv3x512-conv3x512-conv3x512-pool-1024-10
ImageNet	ResNet50, ResNeXt50, GoogLeNet, DenseNet

6.2 Results

6.2.1 Protection. Because the *GSL* vectors are not accessible to the attacker, even if the entire ReRAM content is known, the exact bit streams used in the computation are still hidden from the attacker. Each *GSL* vector has 64bits, the attacker needs to try 2^{64} different combinations to ensure the right one is included. In addition, there are $n = 8$ *GSL* vectors, so the total compute complexity is 2^{67}. Figure 9 shows the accuracy when the attacker guesses a *GSL* vector that is very close to the real *GSL* vector. The first bar is the baseline accuracy when the real *GSL* vector is used. The second bar shows the accuracy if there is only a 1-bit difference between the attacker guessed *GSL* vector and the real *GSL* vector. All of the benchmarks suffer a large accuracy loss. Note that MNIST is a very small dataset, it could be very easily trained to 99.9% accuracy. So a 5% accuracy loss shown in Figure 9 for MNIST already renders significant accuracy degradation. When more bits are guessed incorrectly, the accuracy degradation becomes even larger.

Table 3 shows the effectiveness of SRA's protection on intermediate results. Assuming the target device uses inject pattern A for SC conversion. As can be seen from the table, the accuracy is very close to the baseline (less than 1% accuracy loss). If the model is

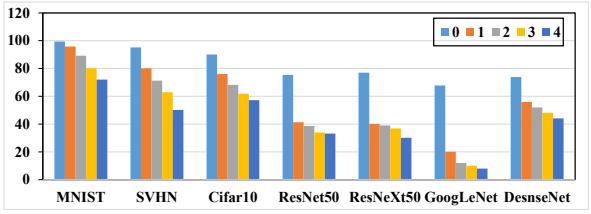


Figure 9: Accuracy with increasing number of *GSL* vectors.

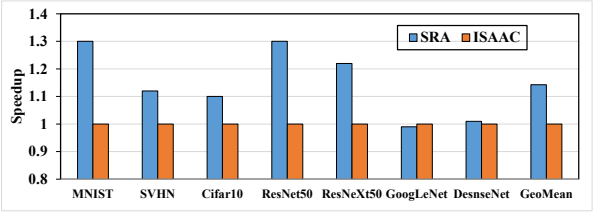


Figure 10: Speedup of SRA over ISAAC.

stolen and applied to a pirate device, the attacker can only use a different inject pattern (assuming inject pattern B). As shown by the fourth column in Table 3, the inference on the pirate device suffers a huge accuracy degradation. The last column shows that the stolen model is also unusable if the attacker uses the binary format weights directly for inference.

Table 3: Inference Accuracy

	Baseline	Inject Pattern A	Inject Pattern B	Binary Weights
MNIST	99.35%	99.36%	14.98%	77.83
SVHN	95.94%	95.23%	67.75%	62.03
Cifar10	90.13%	90.11%	39.47%	52.29
ResNet50	76.13%	75.43%	14.82%	20.8
ResNeXt50	77.61%	77.01%	13.9%	17.84
GoogLeNet	69.77%	67.82%	10.44%	11.81
DenseNet	74.434%	73.86%	20.44%	32.55

6.2.2 Performance. Figure 10 shows the performance of SRA compared to ISAAC. Even if SRA uses multiple *GSL* vectors to reduce the storage overhead caused by the long bit streams in SC format, the total weight storage is still 4× larger than the binary format in the baseline. So more cells need to be activated to perform computation. However, since SRA only activates half of the crossbar at a time, the read speed is much faster than activating the entire crossbar[28]. Also, because every segment of 4 cells is completely isolated from other cells in the crossbar, the sneak current problem is effectively suppressed, reducing the sensing delay of the ADCs. As a result, the overall latency is slightly better than the baseline. On average, there is a 1.14× speedup.

6.2.3 Energy. Figure 11 shows the energy consumption of running the tested benchmarks on SRA normalized to ISAAC. Although each computation needs two ReRAM accesses – one for reading the sign bits and the other for MAC operations, the overall energy consumption is still almost the same or slightly better than the baseline. This is because the SC conversion only reads one row out from the crossbar, its energy is much smaller than a full crossbar read. For MAC operation, SRA consumes less energy because only half of the cells are activated during computation, and the other half is completely shut off by the transistors of each segment. On average, SRA has 9% less energy consumption compared to ISAAC on average.

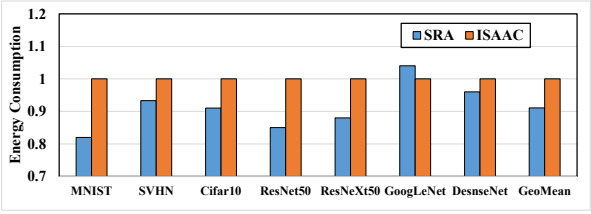


Figure 11: Energy consumption of SRA over ISAAC.

7 CONCLUSION

This paper proposes the first protection approach in the literature for non-volatile crossbar DNN accelerators. In addition to full protection on both weights and intermediate results while they are on-chip, SRA also achieves performance and energy consumption benefits over a conventional non-secure ReRAM based baseline.

REFERENCES

[1] Y. Chen *et al.*, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural network," in *ISCA*, 2016
[2] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014
[3] N. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017
[4] H. Wong *et al.*, "Metal–oxide RRAM," in *Proceedings of IEEE*, 2012
[5] A. Vincent *et al.*, "Spin-transfer torque magnetic memory as a stochastic memristive synapse," in *ISCAS*, 2014
[6] G. Burr *et al.*, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," in *IEEE Transactions on Electron Devices*, 2015
[7] L. Zhao *et al.*, "AEP: An error-bearing neural network accelerator for energy efficiency and model protection," in *ICCAD*, 2017
[8] L. Zhao *et al.*, "SCA: a secure CNN accelerator for both training and inference," in *DAC*, 2020
[9] W. Li *et al.*, "P3M: a PIM-based neural network model protection scheme for deep learning accelerator," in *ASPDAC*, 2019
[10] C. Yeh *et al.*, "Compact one-transistor-N-RRAM array architecture for advanced CMOS technology," in *IEEE Journal of Solid-State Circuits*, 2015
[11] A. Ren *et al.*, "SC-DCNN: Highly-Scalable Deep Convolutional Neural Network using Stochastic Computing," in *ASPLOS*, 2017
[12] S. Li *et al.*, "Scope: A stochastic computing engine for dram-based in-situ accelerator," in *MICRO*, 2018
[13] M. Bojnordi *et al.*, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *HPCA*, 2016
[14] L. Zhao *et al.*, "Flipping Bits to Share Crossbars in ReRAM-Based DNN Accelerator," in *ICCD*, 2021
[15] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016
[16] N. Muralimanohar *et al.*, "CACTI 6.0: A tool to model large caches," *Technical Report*, 2009.
[17] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *TCAD*, 2012.
[18] M. Saberi *et al.*, "Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation ADCs," *TCAS1*, 2011.
[19] Y. LeCun *et al.*, "The mnist database of handwritten digits," http://yann.lecun.com/exdb/mnist/, 2011.
[20] Y. Netzer *et al.*, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
[21] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," *technical report*, 2009.
[22] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.
[23] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998.
[24] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, 2016.
[25] S. Xie *et al.*, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.
[26] C. Szegedy *et al.*, "Going deeper with convolutions," in *CVPR*, 2015.
[27] G. Huang *et al.*, "Densely connected convolutional networks," in *CVPR*, 2017.
[28] T. Yang *et al.*, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *ISCA*, 2019.