WESCO: **W**eight-**e**ncoded Reliability and **S**ecurity **Co**-design for In-memory Computing Systems

Jiangwei Zhang*, Chong Wang*, Yi Cai*, Zhenhua Zhu*, Donald Kline, Jr†, Huazhong Yang*, Yu Wang*
Department of Electronic Engineering, Tsinghua University*
Intel Corporation †
{jwzhang.pitt@gmail.com, yu-wang@mail.tsinghua.edu.cn}

***Abstract*—Non-volatile memory (NVM) based in-memory computing (IMC) systems can avoid expensive data movement by implementing matrix-vector-multiplication calculations in memory, significantly reducing the power consumption and memory bandwidth requirements of deep neural networks (DNNs). Due to the non-volatility and the limited endurance of NVM devices, the system is ideal for low-power and retrain-free applications. However, NVM devices have reliability problems caused by device faults and data security risks due to non-volatility, making the system unreliable and unsecure. We observe that the impact of high-bit faults (HBFs) of quantized weights is far greater than low-bit faults (LBFs) on the classification accuracy of DNNs. Leveraging this observation, this paper proposes a lightweight and efficient co-design of reliability and security for retrain-free IMC systems, called WESCO, that can simultaneously tolerate faults and obfuscate the network. The weight matrices are encoded in row level by swapping the HBFs into LBFs to reduce the impact of faults on network accuracy without retraining; meanwhile, the implementation of our HBF and LBF swapping simultaneously obfuscates the network, so that the models cannot be accurately extracted from the stolen weights. The experimental results demonstrate WESCO can restore the classification accuracy of the DNN models to the baseline level at high fault rate of 5E-3 with a low area overhead of 1.17%, and limit the possibility of attackers stealing the model to infeasible brute force attacks.**

***Index Terms*—DNN, Non-volatile memory (NVM), In-memory computing, Reliability, Security, Fault-Tolerance**

I. Introduction

Smart devices or Internet of things (IoT) nodes are widely used in various fields, and they provide specific functions through Deep Neural Network (DNN) inference. Due to their constrained energy, an energy-efficient implementation of DNN inference is highly preferred. In many important real-world scenarios, retraining is not applicable because of the lack of manpower or training data [1]. What's more, to retrain DNN models requires large-scale calculation and data migration, which is time- and power- consuming. Retrain-free accelerators with in-memory computing (IMC) implementations using non-volatile memory (NVM) such as Resistive Random Access Memory (RRAM) and Ferroelectric Random Access Memory (FeRAM) are promising solutions, which can augment neural network inference with low power and high efficiency. NVM devices are great carriers to perform matrix vector multiplication (MVM)—a dominant operation of neural network computation. Prior to NVM adaption to DNN accelerators, however, there are inherent reliability and security issues which must first be resolved [2], [3].

RRAM devices have potential faults such as manufacturing defects, aging, heat, and random noise interference, which make some units unable to accurately store and calculate, directly resulting in loss of accelerator accuracy when utilized for that purpose [4], [5]. Although DNNs innately have a degree of fault tolerance, the accuracy of DNN will drop rapidly if the fault rate exceeds a certain limit [6]. Take VGG16 as an example, in Figure 1, when the fault rate exceeded 1E-5, the classification accuracy of Origin (without protection) began to deteriorate. This deterioration can be significantly mitigated if we can guarantee protection of the first 4 (H4_correct) or 2 (H2_correct) bits of the weights. The lines for H4_correct and H2_correct remained at the baseline level until the fault rate surpassed 2E-4 and 3E-3, respectively, which were 20× and 300× improvements over the counterpart, while the line for L4_Correct (the last 4 bits protected) did not show any benefit. Therefore, it is clear that the importance of correcting high-bit faults (HBFs) is much higher than that of low-bit faults (LBFs).

The security issues of the DNN accelerator are partly related to the non-volatility of the devices [2]. NVM such as RRAM will not lose data after power is turned off. Although the static power consumption is low, it also brings the security risk of data theft. If an attacker obtains memory hardware, the DNN models may be stolen, causing significant loss of intellectual property. Therefore, it is essential to encrypt the data so that even if the hardware is lost, the model will not be stolen.

This paper endeavors to solve both the hardware faults of NVM devices and the risk of model theft caused by their non-volatility by creating a reliability and security co-design, WESCO, which is short for "Weight-encoded Reliability and Security Co-design". This is the first co-design work towards the two problems to produce robust retrain-free NVM-based IMC systems. Existing methods primarily focus on either solving reliability or security. Our method converts HBFs into LBFs through bit swapping by encoding the weight matrices and simultaneously obfuscates the weight matrices that the accuracies of the models are decreased to an invalid level.

In summary, the main contributions of this paper are summarized as follows:

1) It proposes a new fault tolerance method, called WESCO, according to the asymmetric impact of HBFs and LBFs, which converts HBFs to LBFs by bit swappings and corrects the majority of the faults
2) It simultaneously obfuscates the weight matrices due to

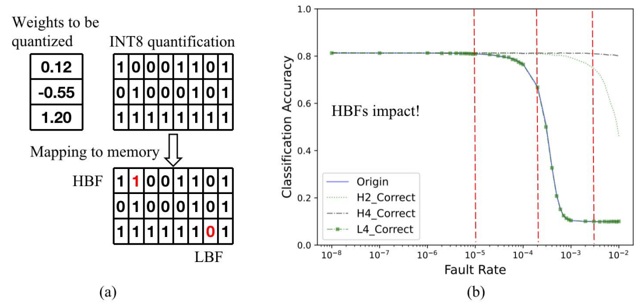


Fig. 1. (a) Hardware Faults in the quantized weights. (b) The impact of HBFs on the classification accuracies of VGG16 at different fault rates.

the swappings of faulty rows, so the model is made secure against physical theft

3) It provides characterizations of the tolerance of stuck-at faults and the obfuscation of the weight matrices of WESCO and conducts an extensive study based on Alexnet, VGG16, and Resnet18, illustrating the significant improvements brought by the proposed strategy

II. Background and Related Work

A. DNNs

Convolutional neural networks (CNNs) are one of the most widely used classes of DNNs. CNNs are mainly composed of convolution (CONV) and fully-connected (FC) layers. The core calculation of these layers is MVM, that is, the dot product of the input data and the weight matrix. The traditional calculation complexity is O(n^2), while Processing-in-memory (PIM) design reduces the complexity to O(1). DNN usually has two stages, training and inference. In the training stage, the network learns certain tasks or patterns from the input datasets; the inference stage is when the results of this training are used for practical applications. We assume that the training stage is done on fault-free memory, a one-time large-scale procedure, and the trained model is deployed to the NVM-based PIM hardware with potential device faults for low-energy inference operations. As the faults in NVM array are randomly distributed and fault pattern of each NVM array is distinct, if each array is customized during training based on its fault pattern, the cost of training will be extremely high.

B. NVM Devices and Security

NVM devices include emerging devices such as RRAM, Phase Change Memory (PCM), and FeRAM. The devices are utilized as high-density crossbars and are commonly used in PIM-based DNN accelerators. NVM devices can be manufactured as single-level cell (SLC) and multi-level cell (MLC). This paper focuses on SLC devices.

MIT tested abandoned hard disks and successfully recovered most of the hard disk data [7]. After stealing the non-volatile device, the attacker can not only obtain the DNN model, which makes it easier to carry out adversarial attacks, but upon learning the model may perform more sophisticated attacks leveraging the knowledge of the layout to cause the outcome of a query to change in a specific way (for example, making a user or competitor come to the wrong conclusion), greatly increasing the security risk. Therefore, NVM-based DNNs clearly need security protection. Apart from model stolen attack, fault injection attacks cause inference errors by injecting faults to reduce network accuracy [2]. Side channel attacks (SCAs) steal information through methods other than looking at the input and output data alone, such as using this information in conjunction with latency, temperature, and a wide variety of other indirect information. In this paper, we focus specifically on security concerns related to physical device theft or direct/indirect observation of data stored in NVM memory. Software threats, transmission-time side channel attacks, and manufacturing-time hardware Trojans are orthogonal to our application of NVM for DNNs.

C. Related Work

There are four main classes of error correction methods for existing DNN accelerators: First is redundant replacement, including row/column-level replacement [8] and Dual/Triple Modular Redundancy (DMR/TMR)) [9]. Fundamentally, this concept is to store duplicates of data, and use the duplicates to correct faulty bits, at the cost of rows or columns, resulting in large area overhead. The second class is to utilize the inherent fault tolerance or redundancy of DNNs to improve the reliability through fault tolerance training or pruning [10]. In the case of high fault rate, each chip has a different fault distribution, the training or pruning overhead is significant and to some extent the improvement is limited [11]. This class of methods requires changes to the DNN models and have high dynamic energy consumption. The third class reduces the impact of faults on network accuracy, including limiting the value range of weights, and intervening and regulating the values that exceed the limit, so that the network accuracy is maintained at a high level [5]. This class of methods have low error correction cost but limited error correction capability, and are only suitable for memory with low fault rates. The fourth class is algorithm-based fault tolerance or error correction coding techniques. Checksum [12] and arithmetic codes [3] are mainly used to detect and correct soft faults at runtime with low fault rate. Traditional memory error correction methods, including Error Correction Codes (ECC) [13] and other NVM-oriented methods can tolerate stuck-at faults [14]–[17]. They are not suitable for in-memory processing and do not fully utilize the fault tolerance of DNN itself. Consequently, at a relative high error rate (1E-5 to 1E-2), existing methods are not yet able to maintain the accuracy of the network without retraining at a relatively low area cost.

For the security problem caused by the theft of non-volatile memory, the existing solutions mainly fall into three categories. First is full encryption, which encrypts all data. Encryption is effective and versatile, but the cost of encryption and decryption is high. For devices with limited endurance, the large number of write operations generated by encryption and decryption accelerates the wear of the device [18]. Second is the partial encryption method, which uses gradient matrix or Hessian matrix to filter out the weights that have a greater impact on the classification accuracy [2]. Encrypting the key weights makes the inference of the encrypted model invalid. However, this category of methods also has the problem of device wear in the position of the key weights. Third is to protect the model information by obfuscating the weight matrix. One method in this category is that RRAM sometimes uses positive and negative arrays to represent the weight matrix. Through row-level random shifting, the positive and negative arrays cannot correspond correctly [19]. This method has only be applied to RRAM with two arrays. The other method in this category is to cut the large weight matrix into small matrices, and randomly shifts the small matrices along the column dimension, so that the multiplication of the input and the weights cannot be corresponded [20]. This method has a limited number of shifting cases. Besides, both methods cannot tolerate any fault for potential faulty NVM array. More importantly, these methods have limited encryption options restricted by the attached multiplexers. If unluckily, these options are leaked, they can no longer protect the model. Our method does not rely on multiplexers to obfuscate the network and has tremendous options.

III. Methodology

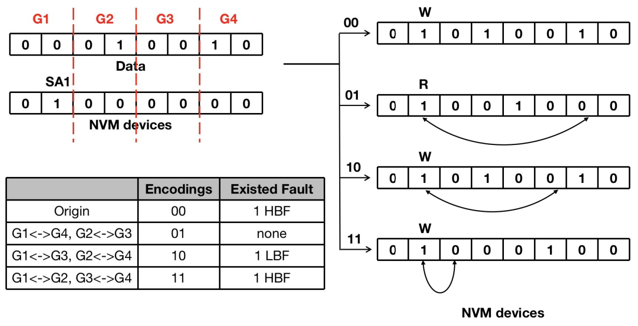
A. Fault tolerance

To tolerate each fault individually is expensive (by recording the address of the fault and the correct data), our method deals with the faults in a row simultaneously. The proposed method not only swaps the high-impact faults in high bits to low-impact low bits to reduce the impact of the faults on network accuracy, but in many cases also tolerates them.

Faults are injected into the NVM array and a certain number of cells are regarded as stuck-at '0' or '1' faults. The quantized weights of pretrained model are written into the NVM array with hard faults. Since they only need to be written once when the model is deployed, in general, there will be no new faults introduced during usage of one particular model. If any, the few faults can be easily detected. If the written data does not match the data in the stuck-at cells, they are Stuck-at Wrongs (SAWs), while if the written data is consistent with the data in the devices, they are Stuck-at Rights (SARs). In addition, for non-permanent faults or "weak" cells discovered after fabrication, we can also treat them as permanent SAWs. The faults are detected by using Read after Write (RAW) technique. SARs faults do not affect the model accuracy, while SAWs do, especially the SAWs in high bits of the weights.

If the swapping transfer SAWs into SARs, when the data bits of the weights are written to different NVM devices, the SAWs are regarded as being corrected. To correct a SAW (stuck-at '0' or '1'), you need to correspond it with a bit valued '0' or '1', and similarly, to keep the correctness of a SAR (stuck-at '0' or '1'), correspond it with the same data or keep unchanged. If faults cannot be corrected by encoding, we have as a final protection the structure of the DNN itself to help protect against faults. For example, VGG16 itself can tolerate the faults when the fault rate is lower than 1E-5.

Swapping needs auxiliary bits each row to record as an encoding. Theoretically, the more encodings, the higher the probability of HBFs being converted to LBFs without new HBFs created. Therefore, as the anticipated fault rate of a weight matrix increases, we need more auxiliary bits (b) to record the encodings. When b=1, there are two encodings that '0' means no swapping and '1' means swapping that the bits in a weight are shifted in '$9 - x$' transformation, where x represents the bit position, marked from the most significant bit (Bit '1') to the least significant bit (Bit '8'). When b=2 or 3, there are four or eight encodings. We will next illustrate the method using b=2 for example.



	Encodings	Existed Fault
Origin	00	1 HBF
G1<->G4, G2<->G3	01	none
G1<->G3, G2<->G4	10	1 LBF
G1<->G2, G3<->G4	11	1 HBF

Fig. 2. An example of WESCO to convert an HBF to an LBF and tolerate the fault with two auxiliary bits per row to record the encodings.

As shown in Figure 2, NVM devices are reset and 8-bit input data ('00010010') is to be written to the devices. Bit '2' is stuck at '1' (SA1), but the data is '0', so that it becomes a SAW and also a HBF that needs to be handled. We partition the weight into four groups, with two bits in each group, numbered G1 to G4. Figure 2 lists the four swapping modes, encodings and fault conditions after swapping. The swapping method is mirror interchange, where G1<->G4 means that G1 and G4 are interchanged, that is, Bit '0' and Bit '1' in G1 are mutually exchanged with Bit '7' and Bit '6' in G4, respectively. Meanwhile, G2<->G3 means that Bit '2' and Bit '3' in G2 are exchanged with Bit '5' and Bit '4' in G4, respectively. The encoding ranges from '00' to '11'. HBF (W1) still exists at encodings '00' and '11', while HBF is converted to LBF (W1) at encoding '10', and HBF is shifted to the low position and corrected at encoding '01'. Therefore, the encodings '01' and '10' swaps the HBF to LBF, while the encoding '01' also tolerates the fault.

When b=3, four additional encodings are created by the parity bit conversion of the four encodings at b=2 in Figure 2. With the eight encodings, any bit can be interchanged to any position, and any bit will not overlap in positions between two encodings. Therefore, faults can search for matching data as far as possible within the weights.

B. Security

Bit swappings of the weights can not only weaken the impact of or correct faults, but can also be utilized to provide security protection. The swappings can encrypt the weight matrix and the key is the encodings. If the encoded matrix is directly used for MVM calculation, the accuracy of DNNs will be greatly influenced, being as low as the level of random chance.

In order to prevent the attacker from obtaining the encodings, the auxiliary bits need to be encrypted and stored in a separate and safe memory, being invisible to users. Since the number of auxiliary bits is not large, during use they can be stored locally in CPU caches or in other memories.

The difficulty of attack is related to the ratio of swapped rows and the number of encodings. The more encodings, the more difficult it is for an attacker to recover the network using the brute-force attack. Theoretically, if the number of rows in a certain weight matrix is n_i and the number of encodings is m_i ($\leq 2^b$), the attacker needs to perform $m_i^{n_i}$ times of MVM calculations to traverse all possible situations of the weight matrix. For the entire DNN model, $\prod_{i=1}^{N_M} m_i^{n_i}$ times of MVM calculations are required, where N_M is the number of weight matrices. Even if b=1, for a large DNN it will be astronomically expensive for the attacker to calculate $\prod_{i=1}^{N_M} 2^{n_i}$ times.

If the attacker could read the values of device, and he or she could also know the locations and faulty types of the faulty devices by probing, the attacker might reverse the encodings of the faulty rows. To reduce this risk, we strategically encode the non-faulty rows by random encodings. Even if all the encodings of the faulty rows are reversed, the encodings of the non-faulty rows can still protect the model from being stolen. If the numbers of faulty rows and non-faulty rows are n_{if} and n_i-n_{if}, respectively, for weight matrix i. It will still be very expensive for the attacker to calculate $\prod_{i=1}^{N_M} {m_i}^{n_i - n_{if}}$ times.

C. In-memory Computing Design

Since the bit swappings may change the order of the bits of the weights in the faulty rows, the MVM calculation cannot be directly implemented. We separate the multiplication of an input vector and a weight matrix into m times of multiplications, that is, each encoding needs one multiplication. Since $m \leq 2^b$ and b=1, 2, 3, the computational complexity is still O(1).

Figure 3 is a demonstration to calculate the sum of the weights in four rows, when b=2. The gray and white bits of the input data are '1' and '0', respectively. The input data (IN) is '1111'. There are four rows in the weight matrix (WM), and each row has only one weight (INT8). The encodings of each row are recorded in auxiliary bits (AUX), including '00', '01' and '11'. We divide IN into IN$_i$ (i=1,2,3), each of which represents the input data of an encoding. IN$_i$ is set that the input data corresponding to the rows of the encoding i is copied from IN, and the remained data are set '0'. In Figure 3, WM contains the weights after swapping. The first and third rows remain unchanged because there is no SAW. The second and fourth rows are swapped to convert the high SAW to the low SAR respectively. Next, each IN$_i$ is multiplied with WM to get Y_i, which is reversed to obtain $Y_i^{'}$ according to AUX, and finally $Y = \sum_{i=1}^{3} Y_i$.

Therefore, the calculation for each encoding requires to set the input data, do matrix multiplication, reverse the swapping, and calculate the summation. Finally, $Y = \sum_{i=1}^{m} Y_i = \sum_{i=1}^{m} \text{IN}_i \cdot \text{WM}$. When $m < 2^b$, there is one or several possible encodings that do not exist in AUX, less times of calculations are required. Hence, if the number of rows of any encoding is very few, we could cut off the encoding to reduce calculation, by distributing the rows to their secondary encodings. In the following section, we analyze the latency, area, and power overhead of WESCO.

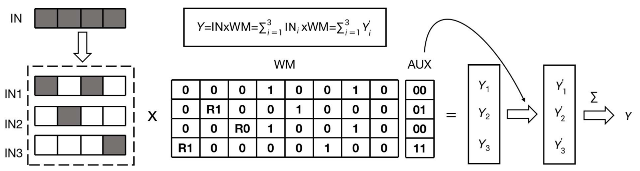


Fig. 3. An example of in-memory computing design of WESCO to perform summation of four weights.

IV. Overhead Analysis

The hardware implementation of in-memory computing commonly divides a large matrix with N rows into multiple small matrices with 8 rows, and then does dot multiplication for each small matrix, and finally accumulates the calculation results [21]. WESCO can easily be applied to the implementation with very low overhead. We can group the rows of one encoding into the same small matrices. If the number of rows of one encoding is less than 8, we can randomly select fault-free rows and encode them using this encoding to make up the difference; if the number of rows of one encoding is greater than 8, we can separate the rows into small matrices, and make up the difference if necessary. Thus, the operations of MVM calculation for each small matrix and the summation of each MVM result are not extra procedure for WESCO.

Besides, to implement WESCO, we need to traverse AUX to search the rows of each encoding and set the input data. And also, after each MVM calculation, we need to remap the swapped weights. These two operations can be hidden in the system level by accomplishing them in parallel with the MVM calculation. As to security purposes, due to the small relative size of the auxiliary bits, they can be cached and quickly fetched for resolving each MVM calculation result.

For the area overhead, as the weights in each row share an encoding, the overhead is low. If each row has M weights, b auxiliary bits, and the weight quantization is q integral bits, the area overhead is $b/(Mq)$. For example, if b=2, M=128, and q=8, the area overhead is only 0.195%.

We assume the number of rows in each weight matrix is constant. When M is increased by c times, the number of matrices decreases by c times. These weight matrices can be calculated in parallel, and the latency is basically no increase. The number of DACs and ADCs will increase by c times because of more matrices, so the area and power consumption of DACs and ADCs will encounter the same times of increase. However, the overhead of ADCs can be partially offset by increasing the number of rows per matrix.

V. Evaluation

We use VGG16, Alexnet, and Resnet18 to verify the effectiveness of WESCO in terms of reliability and security. These models were trained and tested on CIFAR-10 with INT8 quantization. The weights were bit sliced and mapped to the same rows of the NVM array with each bit corresponding to a single-level NVM device. The number of weights M per row in the array is set to 32/64/128. The fault model adopted was consistent with the Ares fault model [4]. The faults (stuck-at '0' and '1') were injected in a randomly uniform distribution. The weights were quantized with the benchmark of the maximum absolute value of the weights in each layer of the DNN models, so the quantized weights were bounded by the maximum value. The faults in the convolutional layers were data-independent, which were injected in the pretrained model, while the faults in the activation layers were data-dependent, which were injected layer by layer during the inference process. The results reported are the average values of 50 experiments, which is reasonable for compute-intensive DNN inference. The primary baseline classification accuracies offered by VGG16, Alexnet, and Resnet18 were 85%, 81%, and 92% without any fault, respectively, while the secondary baseline accuracies offered are the results with HBFs guaranteed protection (H4_Correct) at different fault rates. Thus, at any fault rate, H4_correct actually has half of those faults, since we assume through redundancy or another competing method the high bits have guaranteed protection. We use 'Origin' to refer to the case without any protection. Note that for CIFAR-10, which has 10 classified objects, the classification accuracy 10% means that the network inference is equivalent to random chance, and thus invalid.

To validate WESCO in fault tolerance, we compare with BReLU [11], Ranger [22], and ClipAct [5], which tolerate faults without retraining. BReLU is the state-of-the-art method. Compared to the baseline, we record the fault rate when the accuracy drops by one percent for each method. WESCO has a 1.17% area overhead in maximum, while BReLU, Ranger, and ClipAct have 13%, 10%, and 10% overhead, respectively [11].

Figure 4 shows the classification results of VGG16 when the fault rate ranges from 1E-6 to 1E-2. 'WESCO_3b_32w' refers to WESCO with 3 auxiliary bits and 32 weights per row. The line for Origin keeps constant at baseline level until the fault rate exceeds 1E-5. At the fault rates of 1E-4 and 1E-3, the results of Origin dropped to 72% and 10%, with 13 and 75 percents decrease from the baseline, respectively. At the fault rates of 2E-3, 1E-3, and 1E-3, the lines of BReLU, Ranger, and ClipAct started to decrease, respectively. WESCO_3b_32w, WESCO_3b_64w, and WESCO_3b_128w achieved full protection at the fault rates of 5E-3, 3E-3, and 2E-3, respectively. WESCO achieved up to 2.5x improvement over the state-of-the-art method. It can be seen from Figure 4 that the lines of WESCO_3b_32w and H4_Correct are almost overlapped. H4_Correct had a full protection of the model before the fault rate exceeds 3E-3, while WESCO_3b_32w achieved 1.7x improvement of the point over H4_Correct. This means that, WESCO_3b_32w surpassed the secondary baseline (H4_Correct). At the high fault rate of 1E-2, the results of WESCO_3b_32w and BReLU are 80% and 34%, respectively. The increase in accuracy is 46 percents.

To further validate WESCO, we apply it to Alexnet and Resnet18 when b ranges from 1 to 3. As shown in Figure 5 and Figure 6, the lines of Origin for Alexnet and Resnet18 started to decrease at the fault rates of 1E-5 and 1E-4, respectively, indicating that Alexnet had a comparable ability of fault tolerance to VGG16, but Resnet18 obtained 10x improvement over these two models. WESCO_1b_128w, the weakest version of WESCO shown in the figures, improved 30x and 8x of the starting points over the corresponding Origins, respectively, for Alexnet and Resnet18. The strongest version of WESCO, WESCO_3b_32w, achieved full protection at the fault rates of 5E-3 and 8E-3 for Alexnet and Resnet18, respectively, 2.5x and 1.1x over the comparable points of H4_Correct. At the high fault rate of 1E-2, the results of WESCO_3b_32w for Alexnet and Resnet18 are only 2 and 0.5 percent below the primary baselines, respectively.

Table I lists the number of rows N, the ratio of faulty rows R_F, the ratios of swapped rows (over the number of faulty rows) at b=1 (R_{1b}), b=2 (R_{2b}), and b=3 (R_{3b}), with the fault rate ranging from 1E-6 to 1E-2 and M=32/64/128. N is inversely proportional to M. If the attacker directly uses the encoded model to do inference without decoding, the classification accuracy of the model is around 10%, despite the fault rates and M in the table, which is the invalid level for inference. As the fault rate increased from 1E-6 to 1E-2, R_F increased dramatically from 0.03%, 0.05%, and 0.11% to 92.37%, 99.42%, and 100.00%, respectively, for M=32, 64, and 128. R_{1b}, R_{2b}, and R_{3b} kept almost constant, regardless of M, as the fault rate rises from 1E-6 to 1E-4, and increased moderately as the fault rate reaches 1E-3, and finally jumped to higher levels at the high fault rate of 1E-2. When b increased from 1 to 3, the ratios of swapped rows increased, which

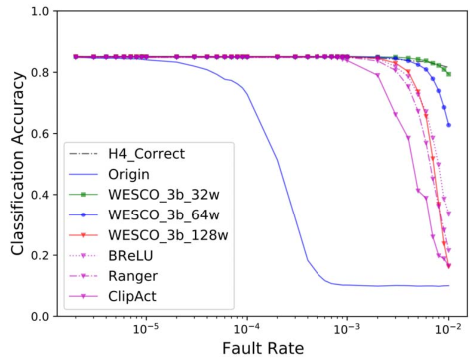


Fig. 4. The impact of faults on the classification accuracies of VGG16 with the protection of various methods. There are 32/64/128 weights per row.

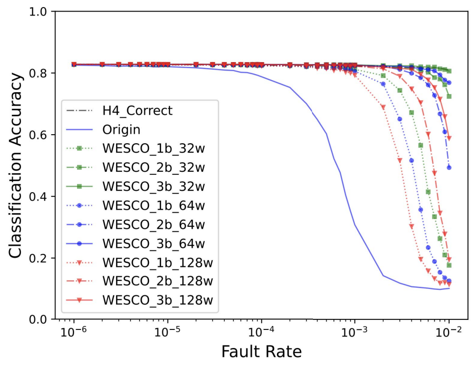


Fig. 5. The impact of faults on the classification accuracies of Alexnet with the protection of various methods. There are 32/64/128 weights per row.

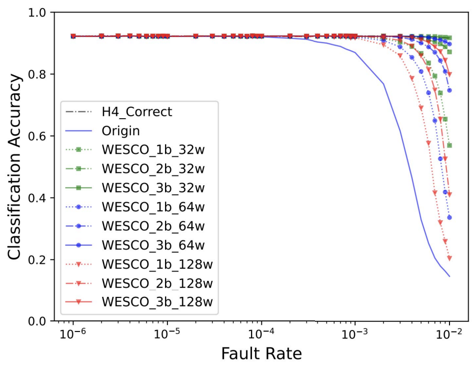


Fig. 6. The impact of faults on the classification accuracies of Resnet18 with the protection of various methods. There are 32/64/128 weights per row.

TABLE I
THE SECURITY ANALYSIS OF VGG16 WITH 32/64/128 WEIGHTS PER ROW AT THE FAULT RATES RANGE FROM 1E-6 TO 1E-2.

M	N	Fault rates	R_F(%)	R_{1b} (%)	R_{2b} (%)	R_{3b} (%)
32	4196954	1E-6	0.03	30.56	44.37	50.18
		1E-5	0.26	30.43	44.20	49.97
		1E-4	2.53	30.65	44.32	50.22
		1E-3	22.60	31.83	46.37	52.46
		1E-2	92.37	41.49	63.19	71.33
64	2098493	1E-6	0.05	30.56	44.38	50.18
		1E-5	0.51	30.44	44.23	50.00
		1E-4	4.99	30.78	44.55	50.47
		1E-3	40.09	33.10	48.6	54.91
		1E-2	99.42	46.87	71.89	81.97
128	1049261	1E-6	0.11	30.56	44.37	50.19
		1E-5	1.02	30.47	44.27	50.04
		1E-4	9.74	31.05	45.02	50.97
		1E-3	64.11	35.53	52.86	59.61
		1E-2	100.00	49.50	75.09	86.81

indicated that more faults were tolerated by swapping.

As for brute force attacks, at the fault rate no less than 1E-6, M=32, 64 and 128, the minimum numbers of calculations are as large as $2^{4196954}$, $2^{2098493}$, and $2^{1049261}$, respectively. The numbers are far greater than $(16!)^{16}$ reported in Ref. [20] and $(16!)^{17}$ in Ref. [19]. Even if the attacker somehow discovers the faulty rows and their encodings, the minimum numbers of calculations are still as large as $2^{2455637}$, $2^{1114929}$, and 2^{415192}, respectively. These numbers are still much larger than the numbers reported in Ref. [20] and Ref. [19].

VI. CONCLUSION

Aiming at the reliability and security problems in the NVM-based in-memory computing systems, this paper proposed to convert the high-bit faults of the quantized weights into low-bit faults by bit swapping, which greatly improved the reliability of DNN inference and protected the models in the case of theft of the NVM device. At fault rate of 5E-3, WESCO restored the classification accuracies of Alexnet, VGG16, and Resnet18 to their baseline levels. At 1E-2, WESCO achieved 46 percents increase in accuracy over the state-of-the-art method with a much smaller area overhead. The protection of the models stolen only required to store a small amount of auxiliary bits in a separate and safe memory. In future work, we will try to optimize the encodings to further improve WESCO's practicability, on the premise of ensuring the effectiveness of the models.

VII. ACKNOWLEDGE

This work was supported by National Natural Science Foundation of China (No. 61832007, U19B2019, 62104128), National Key RD Program of China (No. 2017YFA02077600); Tsinghua EE Xilinx AI Research Fund; Beijing National Research Center for Information Science and Technology (BNRist); Beijing Innovation Center for Future Chips.

REFERENCES

[1] R. Zhao, Y. Hu, J. Dotzel, C. De Sa, and Z. Zhang, "Improving neural network quantization without retraining using outlier channel splitting," *ICML*, pp. 7543–7552, 2019.
[2] Y. Cai, X. Chen, L. Tian, Y. Wang, and H. Yang, "Enabling Secure NVM-Based in-Memory Neural Network Computing by Sparse Fast Gradient Encryption," *IEEE TC*.
[3] B. Feinberg, S. Wang, and E. Ipek, "Making memristive neural network accelerators reliable," *HPCA*, pp. 52–65, IEEE, 2018.
[4] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," *DAC*, pp. 1–6, IEEE, 2018.
[5] L.-H. Hoang, M. A. Hanif, and M. Shafique, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," *DATE*, pp. 1241–1246, IEEE, 2020.
[6] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "RANA: Towards efficient neural acceleration with refresh-optimized embedded DRAM," *ISCA*, pp. 340–352, IEEE, 2018.
[7] P. Roberts, "MIT: Discarded hard drives yield private info," *Computer-World*, Vol. 16, 2003.
[8] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at fault tolerance in RRAM computing systems," *IEEE JETCAS*, Vol. 8, pp. 102–115, 2017.
[9] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM journal of research and development*, Vol. 6, pp. 200–209, 1962.
[10] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training enabled by on-line fault detection for RRAM-based neural computing systems," *IEEE TCAD*, Vol. 38, pp. 1611–1624, 2018.
[11] J. Zhan, R. Sun, W. Jiang, Y. Jiang, X. Yin, and C. Zhuo, "Improving Fault Tolerance for Reliable DNN using Boundary-Aware Activation," *IEEE TCAD*, 2021.
[12] K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, and Z. Chen, "FT-CNN: Algorithm-based fault tolerance for convolutional neural networks," *IEEE TPDS*, Vol. 32, pp. 1677–1689, 2020.
[13] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, Vol. 29, pp. 147–160, 1950.
[14] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," *ACM SIGARCH Computer Architecture News*, Vol. 38, pp. 141–152, ACM, 2010.
[15] J. Zhang, D. Kline, L. Fang, R. Melhem, and A. K. Jones, "Dynamic partitioning to mitigate stuck-at faults in emerging memories," *ICCAD*, pp. 651–658, IEEE, 2017.
[16] J. Zhang, D. Kline, L. Fang, R. Melhem, and A. K. Jones, "Yoda: Judge me by my size, do you?," *ICCD*, pp. 395–398, IEEE, 2017.
[17] D. Kline, J. Zhang, R. Melhem, and A. K. Jones, "Flower and fame: A low overhead bit-level fault-map and fault-tolerance approach for deeply scaled memories," *HPCA*, pp. 356–368, IEEE, 2020.
[18] Y. Cai, Y. Lin, L. Xia, X. Chen, S. Han, Y. Wang, and H. Yang, "Long live TIME: Improving lifetime and security for NVM-based training-in-memory systems," *IEEE TCAD*, Vol. 39, pp. 4707–4720, 2020.
[19] M. Zou, Z. Zhu, Y. Cai, J. Zhou, C. Wang, and Y. Wang, "Security enhancement for rram computing system through obfuscating crossbar row connections," *DATE*, pp. 466–471, IEEE, 2020.
[20] Y. Wang, S. Jin, and T. Li, "A Low Cost Weight Obfuscation Scheme for Security Enhancement of ReRAM Based Neural Network Accelerators," *ASP-DAC*, pp. 499–504, 2021.
[21] C.-X. Xue, J.-M. Hung, H.-Y. Kao, Y.-H. Huang, S.-P. Huang, F.-C. Chang, P. Chen, T.-W. Liu, C.-J. Jhang, C.-I. Su, *et al.*, "A 22nm 4Mb 8b-Precision ReRAM Computing-in-Memory Macro with 11.91 to 195.7 TOPS/W for Tiny AI Edge Devices," *ISSCC*, Vol. 64, pp. 245–247, 2021.
[22] Z. Chen, G. Li, and K. Pattabiraman, "A Low-cost Fault Corrector for Deep Neural Networks through Range Restriction," *DSN*, pp. 1–13, 2021.