Enabling Secure in-Memory Neural Network Computing by Sparse Fast Gradient Encryption

Yi Cai[1], Xiaoming Chen[2], Lu Tian[1], Yu Wang[1], Huazhong Yang[1]
[1]Department of Electronic Engineering, BNRist, Tsinghua University, Beijing, China
[2]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
yu-wang@tsinghua.edu.cn

***Abstract*—Neural network (NN) computing is energy-consuming on traditional computing systems, owing to the inherent memory wall bottleneck of the von Neumann architecture and the Moore's Law being approaching the end. Non-volatile memories (NVMs) have been demonstrated as promising alternatives for constructing computing-in-memory (CiM) systems to accelerate NN computing. However, NVM-based NN computing systems are vulnerable to the confidentiality attacks because the weight parameters persist in memory when the system is powered off, enabling an attacker with physical access to extract the well-trained NN models. The goal of this work is to find a solution for thwarting the confidentiality attacks. We define and model the weight encryption problem. Then we propose an effective framework, containing a sparse fast gradient encryption (SFGE) method and a runtime encryption scheduling (RES) scheme, to guarantee the confidentiality security of NN models with a negligible performance overhead. The experiments demonstrate that only encrypting an extremely small proportion of the weights (e.g., 20 weights per layer in ResNet-101), the NN models can be strictly protected.**

I. Introduction

Deep learning has recently made significant advances in the field of artificial intelligence (AI) [1]. However, the trends toward widening and deepening neural network (NN) architectures have put a tremendous pressure on the computing hardware. Conventional von Neumann architecture is constrained by the inherent memory wall bottleneck, i.e., spending substantial time and energy on moving data between the memory and the processors. Moreover, the Moore's Law is moving towards the end [2], restricting the further optimization of CMOS technologies. Thus, many researchers have turned their attention to the fields of emerging devices and architectures.

Non-volatile memories (NVMs), such as resistive random-access memory (RRAM) and phase change memory (PCM), have emerged as promising alternatives for future NN acceleration [3], [4]. Among the advantages of the NVMs, the non-volatility allows the system to fast restore from hibernation. The high density and low leakage power of the NVMs also provide larger capacity and incur less power consumption. Most importantly, NVMs can construct crossbars to perform matrix-vector multiplications at the location of memory to avoid data moving [5], which is referred to as computing-in-memory (CiM). Many studies have explored CiM-based architectures, especially for NN inference [4] and training [6].

This work was supported by National Key Research and Development Program of China (No. 2017YFA0207600), National Natural Science Foundation of China (No. 61832007, 61622403, 61621091), Beijing National Research Center for Information Science and Technology (BNRist), and Beijing Innovation Center for Future Chips.

Despite the desirable characteristics of NVMs, there are also significant disadvantages and security vulnerabilities in CiM-based NN computing systems. One disadvantage is that the data persist in the memory even when the system is powered off, rendering a security risk of leaking NN models. An attacker with physical access can simply read the memory and extract the weight parameters of the NN models even without powering up the systems [7]. Another disadvantage is that many NVMs have disappointed endurance, typically ranging from 10^6 to 10^{10} [8], [9]. Therefore, the NVM-based systems are vulnerable to frequent and massive write operations. Even under normal use, the lifetime of NVM-based memory and/or computing systems rarely reach the expectation [10], [11].

There are two general encryption approaches to protect the data confidentiality. One approach is *bulk encryption*, which encrypts the entire memory when the systems are powered down and decrypts all when the systems continue working. However, such approach incurs large energy overhead and long encryption/decryption latency. Another approach is *incremental encryption*, which argues that the amount of data involved in an application is much smaller than the entire data set, so only a small percentage of the memory needs to be decrypted when the program runs [7]. However, the computation of NN requires all the weight parameters involved because the inputs are propagated through all the layers. Thus, the system needs to decrypt the entire weights before starting work, and encrypt them again after the work completed. Taking VGG-16 [12] as an example, approximately 138M parameters need to be encrypted/decrypted, introducing large performance overhead. Moreover, both the encryption and decryption incur one write operation in per weight location. Such tremendous writes may also threaten the lifetime of the CiM system. Therefore, there must be a method to substantially reduce the complexity and amount of the weight encryption.

Some researchers have also made efforts on designing encryption techniques to thwart the data confidentiality attacks for NN models. For instance, at the hardware level, P^3M [13] has been proposed based on the physical unclonable functions (PUFs) and processing-in-memory mechanism, which aims to protect the NN models in edge accelerators embedded with eDRAMs. Through their approach, only the authorized device can decrypt the model and make it work normally. However, two drawbacks prevent P^3M from being transferred to the NVM-based NN accelerators: 1) P^3M is dedicated for the eDRAM-based accelerators; and 2) the encryption/decryption still operate on all weights. At the algorithm level, encryption methods such as homomorphic encryption [14] have also been proposed to protect the privacy of NN models. However, these methods are inappropriate for normal NNs and NVM-based accelerators, and are usually with high complexity.

The goal of this work is to find an efficient solution for protecting CiM- and NVM-based NN accelerators from the vulnerability of lingering NN models. The contributions are summarized as follows.

- We analyze the principles of designing the protective solution. To search an approximate optimal solution, we also define and model the weight encryption problem.
- We propose a sparse fast gradient encryption (SFGE) method for efficiently encrypting the weights with negligible overhead to strictly protect the NN models.
- We propose a runtime encryption scheduling (RES) method which disperses the time of encryption/decryption of different layers and pre-decrypts the weights, to ensure the security of NN models at all time.
- We propose an efficient and robust protecting framework for thwarting the NN confidentiality attacks based on SFGE and RES. Thorough experiments have been made and demonstrate only encrypting an extremely small proportion of the weights can prevent the attackers from obtaining the NN models.

II. PRELIMINARIES AND RELATED WORK

A. *NVM-based Neural Computing*

An NVM cell has multiple resistance states, and multiple cells can construct crossbar arrays. By mapping a *matrix* onto the conductance of the NVM cells in the crossbars, and a *vector* onto the input voltages, the NVM crossbar can perform the matrix-vector-multiplications (MVMs) in an extremely high parallelism, without any moving of the matrix data. Assuming the crossbar size as $R \times C$, the relationship of the input voltages and the output currents can be formulated as: $i_{\text{out}}(c) = \sum_{r=1}^{R} g(r, c) \cdot v_{\text{in}}(r)$, where v_{in} denotes the input voltage vector (indexed by $r = 1, 2, ..., R$), i_{out} denotes the output current vector (indexed by $c = 1, 2, ..., C$), $g(r, c)$ denotes the matrix data (i.e., the conductance of the cell) which is in the r^{th} row and c^{th} column of the crossbar. The MVMs dominate the main operations in NN computing because both convolution and fully-connected layers can be decomposed to multiple MVMs. This leads to tremendous opportunities for the NN acceleration by using the NVM crossbars.

B. *Compensation for the NVM Vulnerabilities*

Previous studies have also tried to compensate for the vulnerabilities of the NVMs which mainly include the risk of data leakage and limited endurance. There are two main types of solutions for dealing with the limited endurance problem. One type is reducing the write frequency, such as *Flip-N-write* scheme [15]. The other type is using wear leveling techniques to make the writes uniform across the entire memory, e.g., the *Start-Gap* wear leveling scheme [11]. There are also studies aiming at the NN application, in particular for the NN training. For example, Yi et al. proposes a *structured gradient sparsification (SGS)* scheme which reduces write frequency, together with an *aging-aware row swapping (ARS)* method for the wear-leveling [10]. These approaches provide satisfactory compensation for the limited endurance of NVM under normal use. In addition, there are also approaches that protecting the NVM systems from the malicious wearing-out attacks[16], [11]. So far, the limited endurance is not a major vulnerability that threatens the NVM-based system security.

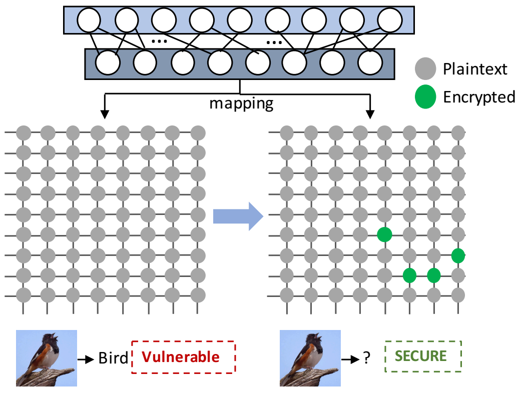


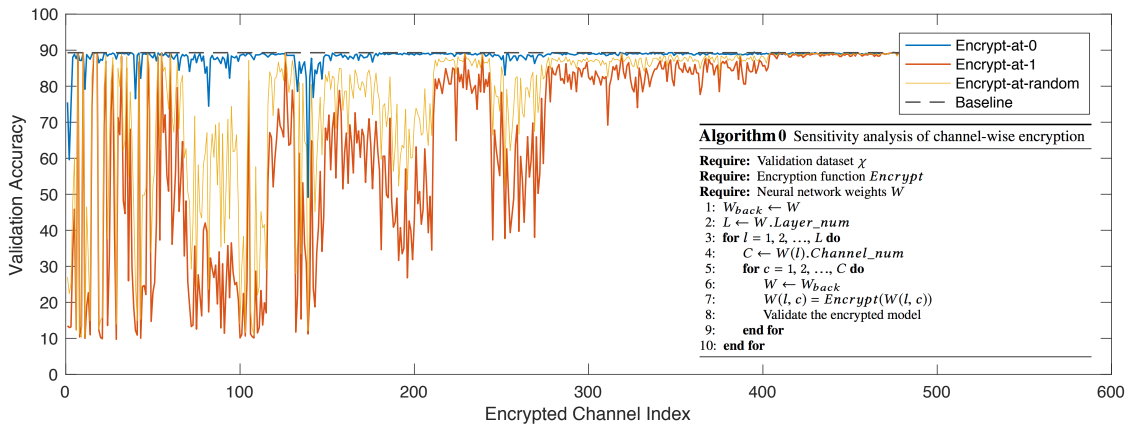
Fig. 1: The goal of the confidentiality protection for NN models. When the weights are mapped on the NVM crossbars in plaintext form, the system will be vulnerable to the confidentiality attacks. Our goal is to encrypt the fewest possible weights to make the NN model misclassified.

There are also many attempts on protecting the lingering confidentiality in the NVMs [7]. However, to our best knowledge, all the prior approaches are proposed for the main memory applications, which are not adapt for the CiM architecture and NN application. The NN computation involves a large amount of weights. Thus, the bulk encryption will incur tremendous performance overhead, especially when deploying large models with heavy weights. Therefore, to implement the encryption at a negligible overhead, the amount of encryption operations should be substantially reduced. As shown in Fig.1, the goal is to encrypt fewest possible weights to make the NN model disabled. Then even if the weights are stolen, the attackers get only a bunch of meaningless numbers.

III. Attack Model

A. *Goal of Security Protection*

The attack considered in this paper is that an attacker with the physical access to a CiM system can extract the NN weights and infer the architecture by bypassing the OS protection and physically reading the memory [17], [7]. With the AI devices becoming increasingly ubiquitous and mobile, the attackers have many opportunities to obtain the physical access to the CiM hardware. The leakage of NN models will lead to serious consequences. On one hand, from a business perspective, the NN model is definitely a core IP. On the other hand, an attacker can launch white-box adversarial attacks to force the NN to make wrong decisions. Therefore, it is necessary to find solutions for protecting the confidential NN models before the CiM-related AI hardware entering the



Algorithm 0 Sensitivity analysis of channel-wise encryption

```
Require: Validation dataset $\chi$
Require: Encryption function $Encrypt$
Require: Neural network weights $W$
 1: $W_{back} \leftarrow W$
 2: $L \leftarrow W.Layer\_num$
 3: for $l = 1, 2, \ldots, L$ do
 4:     $C \leftarrow W(l).Channel\_num$
 5:     for $c = 1, 2, \ldots, C$ do
 6:         $W \leftarrow W_{back}$
 7:         $W(l, c) = Encrypt(W(l, c))$
 8:         Validate the encrypted model
 9:     end for
10: end for
```

Fig. 2: The sensitivity of different channels in ResNet-18 versus the channels indexes. The validation accuracy presents the performance of the encrypted models. The attached algorithm shows the process of the sensitivity analysis.

market. The protection goal is to thwart the threat of leaking the deployed NN models at negligible overhead. Therefore, the principles listed below should be followed.

B. Principles of Security Solution

A satisfactory solution to protecting the NN models in CiM systems should satisfy the following principles:

(1) Functionality. The functionality and performance of the NN models shall be guaranteed under normal use, which means that when performing the computation at some memory locations, all corresponding weights should be decrypted.

(2) Fast Restore. The solution must preserve the instant-on benefit of the non-volatility, i.e., once powered up, the system must fast restore and start working instantly. Since the NN computation always starts from the front layers to the end, it is preferable to encrypt fewest weights in the front layers.

(3) Low Overhead. The solution should not incur large performance overhead. The typical steps of an en/decryption contain: reading the weight from the memory, sending it to the cryptographic engine, executing the en/decryption, and writing the en/decrypted weight back to the memory. Each en/decryption incurs one read, two data moving, and one write. Thus, the amount of encrypted weights should be restricted.

(4) Security at All Time. The solution should keep the system secure at all time, i.e., at any moment, some parts of the NN models are encrypted. Whenever the attackers interrupt the system, they are unable to obtain the entire weights.

(5) Hard to Crack. The solution should be strong enough to prevent being easily cracked. Therefore, two basic requirements must be satisfied. One requirement is that the encrypted elements should be sufficiently concealed to make the encrypted weights undetectable. Another requirement is that the encryption should disable the NN models, and ensure that the attacker cannot reproduce the original models.

IV. Motivational Examples

A. Where and How to Encrypt

As mentioned before, since we cannot encrypt all the weights due to the unacceptable overhead, we must identify the significant weights to keep the NN model secure. A straightforward idea of identifying the significant weights is analyzing the sensitivity of the accuracy to each weight by exhaustive search. Here the *sensitivity* is defined as the influence on the recognition accuracy. However, such exhaustive approach is not practical due to the extremely high complexity and unsatisfying encryption effect.

On one hand, the exhaustive search incurs a high complexity. For example, if we divide the weights into G groups and encrypt them independently to observe their sensitivities, the complexity of the analysis will approach $O(G) \cdot O(Test)$, where $O(Test)$ represents the complexity of a validation round. As the group number G increases, the time required for the search will be substantially enlarged. For instance, to analyze a VGG-16 on the ImageNet dataset, per validation round needs to test 50,000 pictures. Assuming the throughput of the validation system (e.g., GPU) as 200 FPS, per analysis round will consume $250G$ seconds. Even at relatively coarse grouping case that each group contains $1,000$ weights in VGG-16, without considering the overlapping, the analysis will consume 250x138k=34.5 million seconds, approximately 399 days. While if too coarsely grouping the weights, it is quite possible to introduce quantities of ineffective encryption on the insignificant weights. The situation will become much more complicated if considering irregular and cross-layer grouping.

On the other hand, the encryption effectiveness is not satisfactory. We have made experiments of encrypting a ResNet-18 network [18] trained on the CIFAR-10 dataset, as shown in Fig.2. The adopted encryption methods include *encrypt-at-0, 1, and random*, which encrypt the target weights into zero, maximum values, and random values respectively. Due to the sparsity nature of NN, encrypting a single channel at *zero* has little impact on the recognition results. The encrypt-at-1 method can identify the sensitivities of different channels most significantly. However, two problems are raised. First, encrypting an entire channel will be easily detected by the attackers. Second, the analysis does not tell the sensitivity of the back layers, because the channel number increases with the layer depth so that each channel shares limited contribution to

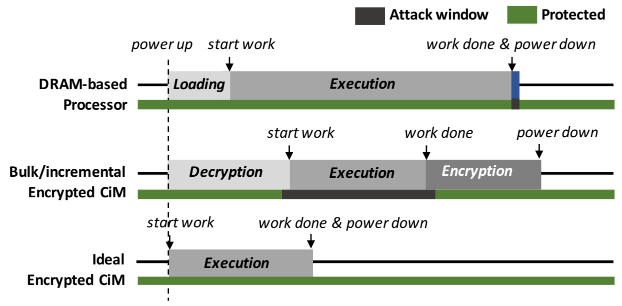


Fig. 3: Contrasting the working order and protection window of DRAM-based processor, bulk/incremental encrypted CiM, and ideal encrypted CiM.

the computation. Therefore, we must find an efficient method to identify the significant weights.

B. When to Encrypt

Another critical problem is when to encrypt the weights. We consider a typical application scenario often encountered in internet of thing (IoT), wearable devices and edge computing with intermittent working mode. When an external signal wakes up the system, it begins to restore and handle the incoming tasks. As shown in Fig.3, in volatile memory-based systems, at most given time, the NN models are protected, because when powered up, there is a software security solution for the data protection; when powered down, the data will not linger. While an encrypted NVM-based system needs to first decrypt the data, then execute the task, and encrypt the data again after the work done. One drawback is that the fast-restore benefit is removed. Another drawback is that there exists an attack window when the system starts working and the weights will remain in plaintext form at this time. Therefore, an attacker still has opportunity to obtain the network weights. An ideal encrypted CiM can keep secure at all time, simultaneously preserve the instant-on property. Due to the staggered computing of different layers, the encryption can be scheduled at run-time to ensure the security at all time.

V. Designing Secure NVM-based NN computing systems

Motivated by the aforementioned observations, we design an efficient solution for protecting the NN models in the NVM-based NN computing systems. The overall framework is shown in Fig.4, which contains two main parts: the sparse fast gradient encryption (SFGE) method for deciding where and how to encrypt, and the runtime encryption scheduling (RES) for deciding when to encrypt. The whole process goes as follows. Before deploying the NN models, we first perform the SFGE to generate the encryption keys, in which the significant weight location and corresponding fast gradient sign are contained. For each NN model, the SFGE operation needs only to be performed once offline. Then, the keys will be kept in the key store (can be double encrypted to enhance the security level). During run-time, the RES module will determine the time of encryption/decryption, take the keys from the key store, and implement the cryptographic operations.

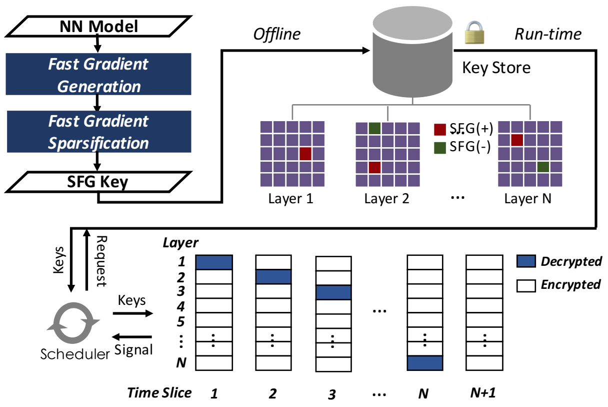


Fig. 4: Overall framework of the encryption solution, which contains two parts: the sparse fast gradient encryption (SFGE) for deciding where and how to encrypt, and the runtime encryption scheduling (RES) for deciding when to encrypt.

A. SFGE: Sparse Fast Gradient Encryption

Inspiration. The fast gradient sign method (FGSM) [19] was first proposed to generate misclassified adversarial examples. An intriguing discovery has been made that a wide variety of NN models are vulnerable to adversarial perturbation on the input because of their linear nature. By adding a small vector whose elements are equal to the sign of the gradient of the cost function with respect to the input, the NN will misclassify the target absolutely. Inspired by that, it is reasonable to argue that the NN models are also vulnerable to the adversarial perturbation added on the weight parameters. Recall that a critical problem of the encryption is to identify the key weights in an NN model, then make small changes on the weights to cause rapid deterioration on the NN classification performance. The fast gradient method can help to find the most significant gradient descent direction.

Another interesting discover which may help us to design the encryption algorithm is the sparse nature of the gradient with respect to the weights. Many studies have explored the gradient sparsification approaches [20] to mitigating the bandwidth requirements in distributed NN training system. For example, deep gradient compression (DGC) [20] demonstrates that one can only preserve approximately 0.1% of the gradients to achieve comparable accuracy with normal training. This enlightens us that the fast gradient can also be sparsified. Recall that the third principle of solution design is not introducing large overhead. Therefore, it is necessary to apply a sparsification on the fast gradient.

Problem Formulation. Let Θ be the weight parameters of an NN model, $\widetilde{\Theta}$ be the perturbation matrix (whose elements are also the encryption keys) added on the original weights, χ be the validation dataset, x be the input to the models sampled from χ, y be the corresponding label associated with x, and $J(\Theta, x, y)$ be the cost function used to train the NN. There are also constraints proposed by the design principles. First, the amount of weight encryption should be within an acceptable boundary. Here we denote the encryption amount as N. We set a selection matrix $Mask$ which contains only 0 and 1, to generate the sparse encryption matrix $\widetilde{\Theta}$. Thus, the number of 1s in $Mask$ should be less than N. Second, the modified weights should not significantly change the distribution of the weights, otherwise there will be outliers which can be easily detected. Therefore, the values of $\widetilde{\Theta}$ should be constrained, and we assume the max perturbation intensity as ϵ. In summary, since our goal is to find the optimal $\widetilde{\Theta}$ to degrade network performance, the encryption problem be modeled as below:

$$\begin{aligned} \max \quad & \sum_{(x,y)\in\chi} J(\Theta + \widetilde{\Theta}, x, y) \\ s.t. \quad & \begin{cases} \widetilde{\Theta} = \Theta^* \odot Mask \\ \mathbb{1}(Mask) \le N \\ \max(|\widetilde{\Theta}|) \le \epsilon \end{cases} \end{aligned} \tag{1}$$

Fast Gradient. Due to the black-box nature of the NN, it is of great difficulty to find an optimal solution for the above optimization problem. Therefore, we give an approximate solution for the problem. In the NN optimization, the most utilized optimization method is gradient backpropagation. The fast gradient with respect to the weights can be obtained by the following equation:

$$\Theta^* = \sum_{(x,y)\in\chi} \bigtriangledown_\Theta J(\Theta, x, y) \tag{2}$$

However, the fast gradient Θ^* is still dense. Since there is an encryption amount constraint in the optimization problem, we need further sparsify the gradients to preserve a small portion of them.

Sparsification. A critical problem of sparsification is how to find the significant gradient that impacts the performance most. Because the partial derivatives for some variable contained in Θ is the rate of change of the function $J(\Theta)$ along the direction, the magnitude of the partial gradient can reflect the descent speed of the cost function along the corresponding variables. Therefore, preserving the gradients with the largest magnitudes can enable a sparse gradient to enlarge the loss. We sort the fast gradients by their absolute values, and preserve the top-N for each layer. Let thr be the threshold of the top-N gradients. At final, to reduce the complexity of the keys, we only preserve the sign of corresponding fast gradient. Therefore, the fast gradients preserved becomes:

$$Mask \leftarrow |\Theta^*| \ge thr \tag{3}$$

$$\widetilde{\Theta} = \epsilon \cdot \text{sign}\big(\big(\sum_{(x,y)\in\chi} \bigtriangledown_\Theta J(\Theta, x, y)\big) \odot Mask\big) \tag{4}$$

Opposite to the normal neural network training which aims at minimizing the loss function, the encryption goal is to enlarge the loss to make the NN misclassified. Therefore, we add the generated sparse fast gradient on the vanilla parameters, then the encryption will be done. We refer to this method as the "sparse fast gradient encryption" (SFGE). Algorithm 1 concludes the overall algorithm and process.

The Composition of the SFGE Keys. The keys of the encryption are composed of two parts: the encrypted location and the encrypted sign. Each key requires $\lceil \log_2(L) \rceil + \lceil \log_2(M) \rceil + 1$ bits to record the encryption information, where the L represents the layer number of the NN model, the M represents the number of the weights in this layer, and 1 bit for indicating the encryption direction of the weight (+ or -).

Algorithm 1 Sparse Fast Gradient Encryption

```
Input: Validation dataset $\chi$ and size $b$
Input: Neural network weights $\Theta$ ($L$ layers)
Input: Cost function: $J(\Theta, x, y)$ (cross entropy loss)
Input: Constraints: encryption amount per layer $N$, encryption intensity $\epsilon$
1: $\Theta^* \leftarrow 0$
2: for $(x, y)$ in enumerate($\chi$) do
3:     $\Theta^* \leftarrow \Theta^* + \bigtriangledown_\Theta J(\Theta, x, y)$
4: end for
5: for $\Theta^*_i$ in enumerate($\Theta^*$) ($i = 1, 2, ..., L$) do
6:     Select threshold : $thr_i \leftarrow$ top $N$ of $|\Theta^*_i|$
7:     $Mask \leftarrow |\Theta^*_i| \ge thr_i$
8:     $\widetilde{\Theta_i} \leftarrow \epsilon \cdot \text{sign}(\Theta^*_i \odot Mask)$
9: end for
10: for $\Theta_i$ in enumerate($\Theta$) ($i = 1, 2, ..., L$) do
11:     $\Theta_i \leftarrow \Theta_i + \widetilde{\Theta_i}$
12: end for
```

Decryption. The only operation introduced by the encryption is the addition on the weights. Therefore, the decryption only needs to add a negative item of the sparse fast gradient key on the encrypted weights, then the NN model will work normally. Thus, the decryption keys can be obtained by simply flipping the last bit (sign bit) of the encryption keys.

Complexity Analysis. Since the SFGE only needs to perform once for each NN model, the complexity will be $O(Test) + O(Sort)$, where $O(Test)$ refers to the complexity of one validation round which has already been defined before, and $O(Sort)$ represents the complexity of the sorting operations to select the gradients with largest absolute values. Concurrently, the optimal solution for selecting top-N gradients from M ones has a complexity of $O(M \log_2(N))$.

Trade-offs. There are two constraints that need to be considered. One constraint is the encryption budget N, i.e., the max number of weights that are allowed to be encrypted. Another constraint is the perturbation intensity limitation, which should not exceed an ϵ to enhance the concealment. There exist trade-offs for balancing the overhead and the encryption effectiveness. An increasing N will incur more overhead, because each en/decryption needs to write on the corresponding weight location. While intuitively, encrypting more weights certainly results in a higher security level. Concurrently, the encryption intensity ϵ also affect the encryption effectiveness significantly. Larger ϵ has greater impact on the performance, while it also increases the probability of being detected because the encrypted weights will exceed the original weight distribution range and become outliers. Therefore, ϵ must be carefully designed based on the actual weight distribution.

B. RES: Runtime Encryption Scheduling

A runtime encryption scheduling is highly demanded to keep the CiM system secure at any given time. The conventional way, which decrypts before starting work and encrypts after ending the work, has security holes that the attackers still have opportunities to steal the model by interrupting the system during run-time. As is known that the NN computing always starts from the first layer, and end in the last layer.

There are dependencies between the layers, i.e., the input of a layer is the output of its previous layer. Commonly-seen computation scheduling among the layers includes the layer-by-layer scheduling and cross-layer co-scheduling.

The layer-by-layer scheduling performs the computation of each layer in sequence, i.e., a layer starts computing when its previous layer has finished the computation. Such working order provides much convenience for the runtime encryption scheduling. In this scenario, the encryption can also be simply done layer-by-layer. The whole process is shown as Fig.4, the layer will only be decrypted when the program comes, and be encrypted after the work has been done. Therefore, at any time, there is only one single layer remaining in plaintext form. In addition, pre-decryption can be performed to hide the decryption latency during run-time.

Another commonly-seen method is cross-layer scheduling, which fully utilizes the parallelism across the layers [21]. There exist a parallel potential to accelerate the processing, because sliding windows are used to convolve the feature maps. Hence, a layer can start computing when fetching a window of the outputs from the previous layer. Although the parallelism of different layers can be exploited, their computation time slices are always staggered due to the dependencies. Therefore, we can profile the working and idle cycles, then fully utilize the idle cycles to perform the cryptographic operations and ensure the security. While we only consider the layer-by-layer scheduling in the following experiments.

Discussion. RES will incur write operations for the en/decryption in every inference round. This raises concerns about the lifetime of the systems. Because the SFGE keys are fixed, frequently operating on the same cells will certainly wear out them. Therefore, two solutions can help overcome this problem: 1) applying the RES only in the intermittent working mode with infrequent activities, such as the energy-harvesting edge devices or some embedded applications (e.g., face identification module in phones); and 2) applying wear-leveling techniques to uniform the writes across the whole memory. It is not difficult to design the wear-leveling strategy because the writing behavior incurred by RES is totally predictable.

VI. Experiments

A. Experiment Settings

We investigate the accuracy influence and protection effectiveness of our solution. The experiments are constructed on the ResNet [18] (with 18, 50, and 101 layers) and VGG-16 models [12] with the ImageNet dataset. The evaluation metrics include: 1) the accuracy influence; 2) the concealment and robustness; and 3) the latency overhead. There are two main parameters involved in the experiments: the encryption amount per layer N and the encryption intensity ϵ added on the weights. Considering that the number of weights in the front layer is usually smaller than the back, we set the encryption amount as $\min(0.1\% \times M, N)$, where M represents the number of weights in corresponding layer.

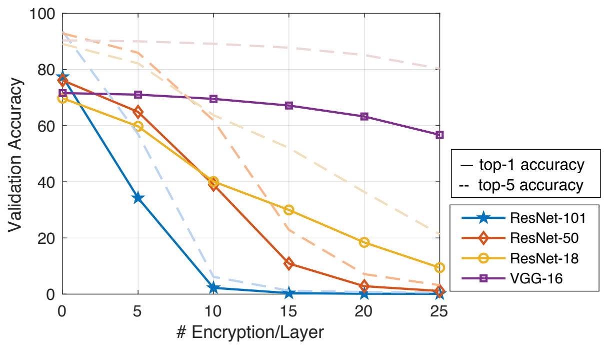


Fig. 5: The validation accuracy of the encrypted NN models versus N ($\epsilon = 0.1$).The accuracy first declines as N increases, then will saturate when reach a turning point. The declining trend also becomes faster when the NN depth increases. Moreover, VGG-16 demonstrates much better adaptability under encryption than the ResNets due to the heavy weights.

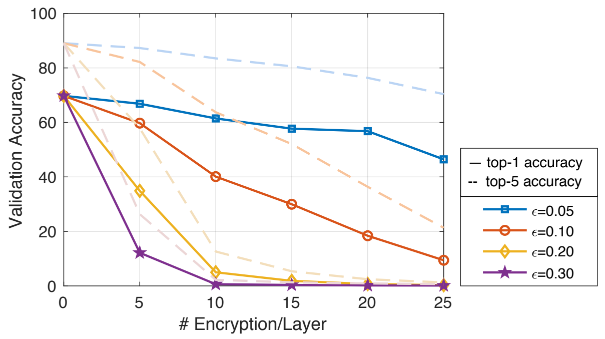


Fig. 6: The validation accuracy of the encrypted ResNet-18 models versus N under different intensity ϵ. As the ϵ increases, the accuracy decline trend becomes faster.

B. Accuracy Influence of SFGE

Encrypting the NN models by SFGE has a significant impact on the accuracy performance. We have evaluated the impact of N and ϵ respectively. Recall that our goal is to destroy the classification ability of the NN models, more influence on the accuracy indicates more effective encryption.

Fig.5 shows the validation accuracy of the NN models versus the encryption amount per layer N. Four interesting conclusions can be figured out from the results. First, the accuracy influence increases with N. This is a predictable conclusion because with encryption amount increases, the encrypted weights will increasingly vary from the original weights. Hence, more computational errors will be introduced. Second, there exists a turning point on the accuracy curves. For example, in the curve of ResNet-101, when N approaches 15, continuing to enlarge N will not be profitable because the performance has been already fully deteriorated. Third, the trend of accuracy dropping is closely related to the NN depth. See the results of ResNet-18, 50, and 101, the ResNet-101 demonstrates the fastest accuracy decline trend. And the decline speed becomes slower when the layer number decreases. A reasonable explanation is that the errors caused by the encryption will grow when propagating through the

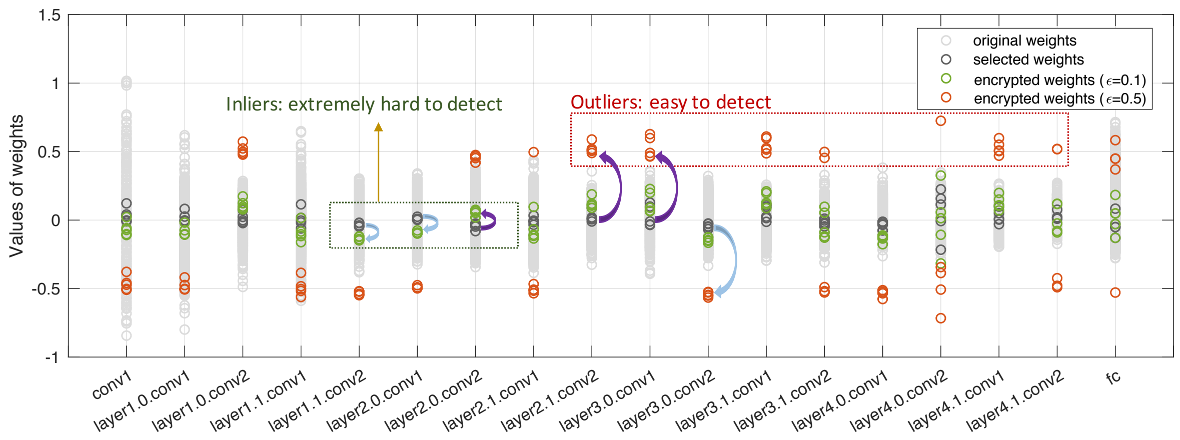


Fig. 7: The weight distribution of ResNet-18 model, in which each weight is figured as a point. We select the top-5 encrypted weights as examples. When $\epsilon = 0.1$, the encrypted weights still fall in the original range and will be extremely hard to be detected; when $\epsilon = 0.5$, many encrypted weights will jump out the normal range and become outliers, increasing the risk of being detected.

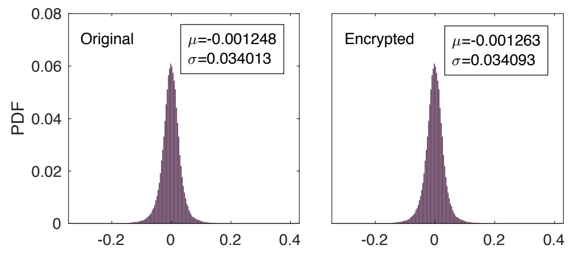


Fig. 8: The weight distribution of layer2.0.conv2 in ResNet-18 before and after the encryption. The encryption configurations are N=20, ϵ=0.2. The mean and variance of the weight distribution remains insignificantly changed after encryption.

layers. Therefore, deeper NNs are influenced more because the errors will be explosively accumulated. Forth, heavy model shows less sensitivity to the encrypting perturbation. ResNet-18 and VGG-16 have similar layer numbers, while VGG-16 shows much better adaptability than ResNet-18. This may be resulted from the $10\times$ heavier weights of VGG-16 than ResNet-18. In this scenario, increasing N or the intensity ϵ can improve the encryption effects.

Fig.6 shows the validation accuracy of encrypted ResNet-18 versus the encryption intensity ϵ. The performance of encrypted models with larger ϵ degrades more quickly than the ones with smaller ones. However, there still exist an intensity limit to enhance the encryption concealment, as will be further discussed below.

C. Security Analysis

SFGE encrypts the weight parameters of neural network models by using the generated SFG keys to protect the well-trained models from the confidentiality attacks. To analyze the security level, we consider four aspects as following.

Concealment of the Encrypted Weights. The SFG keys should be tightly concealed to make the encrypted weights undetectable. The concealment is crucial because once the attackers know the exact locations of encrypted weights, they can exhaustively crack the encryption with a very low complexity. An important indicator of the concealment is that the values of encrypted weights should be within the distribution range of the original weights. We show the weights of ResNet-18 in Fig.7, in which each weight is figured as one point. With a small encryption intensity, such as $\epsilon = 0.1$, the encrypted weights will still fall within the original range. It is almost impossible for the attackers to detect the encrypted weights from the chaotic weight parameters. However, while larger ϵ brings better encryption effect, it also increases the risk of being detected. As shown in Fig.7, when ϵ reaches 0.5, many encrypted weights will jump out the normal range and become outliers. Thus, the intensity ϵ should lie in a reasonable range.

Impact on the Statistical Distribution. A robust encryption requires the weight distribution remaining insignificant changed. We draw the statistical probability distribution of the original and encrypted weights of layer2.0.conv2 in ResNet-18, as shown in Fig.8. there are extremely small fluctuations on the mean μ and variance σ. Moreover, we also calculate the norm squared difference of the original distribution histogram and the encrypted one, which reaches only 5.43×10^{-5}. Therefore, an attacker cannot distinguish the encrypted weights by observing the distribution difference.

Recall of the Encrypted Weights. Another indicator of the concealment is the *recall* of the the encrypted weights. The "recall" is defined as the rate of searched encrypted weights when performing the fast gradient generation again on the encrypted model. Besides, we define top-$100(1000)$ recall which limited the search range to the weights with top-$100(1000)$ largest gradients, because it will be meaningless to continue enlarging the search space as the search complexity in top-1000 has already approached $\binom{1000}{20} \approx 3.4 \times 10^{41}$ (taking ResNet-101 as an example). This concern is raised based on the consistency of the gradient that may make the encrypted model still sensitive to the same weights, so the attackers may collide with the encrypted weights by performing the

TABLE I: Experimental Results of the Encryption for Different Neural Network Models.

Model	Dataset	Classification Accuracy				Encryption Config.	FGSM Adv. (top-1) (Intensity: 0.05)		Mean Recall	
		Encrypted		Baseline						
		Top-1	Top-5	Top-1	Top-5		Encrypted	Plaintext	top-100	top-1000
ResNet-18	ImageNet	0.704% (-69.05%)	2.452% (-86.62%)	69.75%	89.07%	N=20, ϵ=0.2	46.34%	10.47%	0.27%	9.44%
ResNet-50	ImageNet	0.438% (-75.69%)	1.540% (-91.32%)	76.13%	92.86%	N=30, ϵ=0.1	58.74%	21.27%	4.2%	13.0%
ResNet-101	ImageNet	0.144% (-77.24%)	0.758% (-92.78%)	77.38%	93.54%	N=20, ϵ=0.1	60.49%	24.39%	4.2%	14.3%
VGG-16	ImageNet	0.818% (-70.77%)	3.478% (-86.90%)	71.59%	90.38%	N=30, ϵ=0.2	37.59%	11.66%	16.4%	40.8%

fast gradient generation again. The mean recall represents the mean value of the recall rates of all layers. As shown in Table I, the recall of ResNet models are extremely low. Although the VGG-16 shows much larger recall, it is difficult to restore the vanilla weights. Thus, it is almost impossible to re-generate the same gradient keys through the encrypted models.

Defence against Adversarial Examples. An important goal of our protection is to defend the white-box adversarial attacks. Thus, we evaluate the defence effectiveness against the adversarial examples respectively generated based on the encrypted weights and the original weights by using the FGSM. The intensity we apply in the attacks is set as 0.05. As shown in Table I, the NN models are vulnerable to the adversarial examples generated by performing FGSM on the weights in plaintext form. The situation will be greatly improved under the adversarial examples generated based on the encrypted weights. While the performance still degrades, which mainly results from two aspects: 1) the transfer ability of the adversarial examples; and 2) the partially preserved NN characteristic. Therefore, the white-box adversarial attacks can be well defended under the SFGE.

D. Impact on the Latency

Considering the layer-by-layer scheduling, the latency is mainly introduced by the decryption of the first layer in NN when the system starts, because RES will perform the decryption of the following layers during runtime to hide the addition latency. As presented in NVSim [22], the write latency of PCM and RRAM achieves 416.2ns and 100.6ns respectively. The decryption only incurs one write on each encrypted weight, therefore the overall decryption latency will be expected to reach $416.2N$ns (PCM-based) and $100.6N$ns (RRAM-based) respectively. Taking the ResNet-101 as an example, the encryption amount of the first layer is $\min$(7x7x3x64x0.1%, 20)$\approx$10. Thus, the latency will be 4.16μs and 1.01μs respectively, which is negligible in the inference process.

VII. Conclusion

We have modeled the NN encryption problem and presented an efficient protecting solution to thwart the confidentiality attacks which threatens the privacy of the well-trained NN models deployed in CiM- and NVM-based computing systems. An efficient framework has been proposed based on the SFGE method for efficient encryption of the weights and the RES scheme for the runtime scheduling of the weights encryption. Experimental results have demonstrated the effectiveness and robustness of our methods.

References

[1] Y. LeCun *et al.*, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] M. M. Waldrop, “The chips are down for moore’s law,” *Nature News*, vol. 530, no. 7589, p. 144, 2016.

[3] S. Ambrogio *et al.*, “Equivalent-accuracy accelerated neural-network training using analogue memory,” *Nature*, vol. 558, no. 7708, p. 60, 2018.

[4] P. Chi *et al.*, “Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory,” in *ISCA*, vol. 43, 2016.

[5] L. Xia *et al.*, “Technological exploration of rram crossbar array for matrix-vector multiplication,” *Journal of Computer Science and Technology*, vol. 31, no. 1, pp. 3–19, 2016.

[6] M. Cheng *et al.*, “Time: A training-in-memory architecture for memristor-based deep neural networks,” in *DAC*, 2017, p. 26.

[7] S. Chhabra *et al.*, “i-nvmm: a secure non-volatile main memory system with incremental encryption,” in *International Symposium on Computer Architecture*, 2011.

[8] K. Beckmann *et al.*, “Nanoscale hafnium oxide rram devices exhibit pulse dependent behavior and multi-level resistance capability,” *Mrs Advances*, vol. 1, pp. 1–6, 2016.

[9] C. H. Cheng *et al.*, “Novel ultra-low power rram with good endurance and retention,” in *VLSI Technology*, 2010, pp. 85–86.

[10] Y. Cai *et al.*, “Long live time: improving lifetime for training-in-memory engines by structured gradient sparsification,” in *DAC*, 2018, p. 107.

[11] M. Qureshi *et al.*, “Enhancing lifetime and security of phase change memories via start-gap wear leveling,” in *MICRO-42*, pp. 14–23.

[12] K. Simonyan *et al.*, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[13] W. Li *et al.*, “P^{3}m: a pim-based neural network model protection scheme for deep learning accelerator,” in *ASPDAC*, 2019, pp. 633–638.

[14] C. Orlandi *et al.*, “Oblivious neural network computing via homomorphic encryption,” *EURASIP Journal on Information Security*, vol. 2007, no. 1, p. 037343, 2007.

[15] S. Cho *et al.*, “Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance,” in *IEEE/ACM International Symposium on Microarchitecture*, 2009.

[16] N. H. Seong *et al.*, “Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping,” *Acm Sigarch Computer Architecture News*, vol. 38, no. 3, pp. 383–394, 2010.

[17] W. Hua *et al.*, “Reverse engineering convolutional neural networks through side-channel information leaks,” 2018, pp. 1–6.

[18] K. He *et al.*, “Deep residual learning for image recognition,” in *CVPR*, 2016.

[19] I. J. Goodfellow *et al.*, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.

[20] Y. Lin, H. Song, H. Mao, W. Yu, and W. J. Dally, “Deep gradient compression: Reducing the communication bandwidth for distributed training,” 2017.

[21] T. Tang *et al.*, “Binary convolutional neural network on rram,” in *ASP-DAC*, 2017.

[22] X. Dong *et al.*, “Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory,” *TCAD*, vol. 31, no. 7, pp. 994–1007, 2012.