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

Yi Cai, Xiaoming Chen, Member, IEEE, Lu Tian, Yu Wang, Senior Member, IEEE, and Huazhong Yang, Fellow, IEEE

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 adversary with physical access to extract the well-trained NN models. The goal of this article 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. Moreover, we improve the SFGE method by incrementally generating the encryption keys. Additionally, we provide variants of the encryption method to better fit quantized models and various mapping strategies. 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.

Index Terms—Non-volatile memory (NVM), compute-in-memory (CIM), neural network, security, encryption

1 INTRODUCTION

Deep learning has recently made significant advances in the field of artificial intelligence (AI) [1]. The growing computing capability gives more opportunities for the development of neural networks (NNs). 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), e.g., resistive random-access memory (RRAM) and phase change memory (PCM), have emerged as promising alternatives for constructing future NN accelerators [3], [4]. Among the advantages of the NVMs, the non-volatility allows the system to fast restore from hibernation and tolerate power failures. 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], referred to as computing-in-memory (CIM). Many studies have explored CIM-based architectures, especially for both NN inference [4] and training [6].

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 adversary with physical access to the devices 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].

The risk of confidentiality leakage motivates the data encryption. There are two general encryption scheduling approaches to protect the 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, so only a small percentage of the memory needs to be decrypted when the program runs, and most inert data can be kept encrypted [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 network [12] as an example, approximately 138M parameters need to be encrypted/decrypted. Because each encryption or decryption will write the NVM cells to tune the conductance to change the stored weight values, bulk encryption on all weights incurs intensive write operations. Such tremendous writes are unacceptable and also challenge the endurance. Therefore, there must be a solution to substantially reduce the complexity and amount of the weight encryption.

Some researchers have designed encryption techniques to thwart the data confidentiality attacks for NN weights. For instance, at the hardware level, P4M [13] has been proposed based on the physical unclonable functions (PUFs), 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 P4M from being transferred to the NVM-based NN accelerators: 1) P4M 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 optimal solutions, we also define and model the weight encryption problem.
- We propose a sparse fast gradient encryption (SFGE) method for encrypting the weights with negligible overhead to strictly protect the confidentiality.
- We introduce incremental SFGE (i-SFGE) to achieve more efficient encryption. We also specify the approach and provide variants for adapting the quantized models and different mapping strategies.
- We propose a runtime encryption scheduling (RES) method that disperses the time of encryption/decryption of different layers, to ensure the security of NN models at all time and hide the latency.
- 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.

2 Preliminaries and Related Work

2.1 NVM-Based Neural Computing

An NVM cell has multiple conductance levels, and multiple cells can construct crossbar arrays. As shown in Fig. 1, by mapping a matrix onto the conductance of the cells in the crossbars, and a vector onto the input voltages, the 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 $R \times C$, according to the Ohm’s Law and the Kirchhoff’s Current Law, the relationship of the input voltages and the output currents can be formulated as: $i_{out}(c) = \sum_{r=1}^{R} g(r, c) \cdot v_{in}(r)$, where $v_{in}$ denotes the input voltage vector (indexed by $r = 1, 2, \ldots, R$), $i_{out}$ denotes the output current vector (indexed by $c = 1, 2, \ldots, C$), $g(r, c)$ denotes the matrix data (i.e., the conductance of the cell) which is located in the $r$th row and $c$th column of the crossbar [5].

The MVMs dominate the main operations in NN computing because both convolution and fully-connected layers can be decomposed to multiple MVMs. For fully-connected layers, the computation is exactly an MVM computation. For convolution layers, the weight kernel can be unfolded and the computation can be decomposed as multiple MVMs. Through mapping the weights on the conductance of the cells and the feature maps on the input voltages, the NVM crossbars can efficiently implement the operations in NN. This leads to tremendous opportunities for the NN acceleration by using the NVM crossbars.

2.2 AI Hardware Attacks

With AI’s landing, the security of AI has attracted increasing attention, especially in safety-critical applications, e.g., autonomous driving. In the algorithm side, numerous studies have explored the adversarial examples to fool NNs and also discussed corresponding countermeasures [15], [16]. In the hardware side, there are also studies that discover vulnerabilities in AI hardware, mainly including two categories of threat models: integrity attack and confidentiality attack.

The integrity attacks aim at undermining the integrity of the deployed models and cause them unavailable. Hardware Trojans can be injected into the hardware to achieve this goal. Hu-fu [17] injects hidden neurons in the network which works as a backdoor. When triggering the backdoor, the hidden neurons will be activated and the NN models will output wrong predictions. Similarly, [18] designs a memory Trojan and a trigger based on the...
input images. When the trigger is detected, the memory controller will activate the payload and produce wrong outputs. Meanwhile, fault injections can be utilized to make the NN models misclassified by injecting faults into the memory by laser beam or row hammer attacks [19].

The confidentiality attacks aim at extracting the deployed NN models on a variety of hardware accelerators. Side channel attacks (SCAs) are frequently utilized to obtain the NN architectures. For example, Hua et al. [20] utilizes the memory access pattern during the NN inference to reveal the network structures. DeepSniffer [21] is the first work that explicitly and quantitatively evaluates model extraction. It has been demonstrated in the GPU platforms that significantly boosts adversarial attack effectiveness, sounding alarms for model protection. Similar approaches have also emerged to reverse the NN architectures by counting the GEMM calls via cache SCA [22], observing the patterns and timing of the operations [23], etc.

The confidentiality attacks can also be conducted by exploiting the data persistence. In 2003, the MIT researchers acquired 158 discarded hard disks from eBay. They successfully uncovered old data from 117 out of the 158 disks [24]. Later, another group of researchers found that through cold boot, the data stored in DRAM could also retain for several minutes. And within such a short time, the attackers would be able to dump the DRAM and extract the private information [25]. These studies demonstrate that data remanence will lead to severe security vulnerabilities.

Protecting the confidentiality of neural network models is of paramount importance, as the leakage of NN models will lead to severe consequences. First, from a business perspective, the NN models are definitely treated as core intellectual properties for the companies who run the algorithms as key products. If the deployed NN models are cracked, the attackers will be able to duplicate the models without authorization, which greatly harms the benefits of model providers. Second, the well-trained models are usually trained on private and sensitive data. Therefore, confidential information may be encoded into the NN models and the attackers may reverse sensitive information if they obtain the models. Third, adversarial example attacks [26] have been demonstrated as a huge risk to the neural network security. If the attackers get the exact architectures and parameters of the NN models, they can launch white-box adversarial attacks to force the NN to make wrong decisions, which is much easier than black-box attacks. Therefore, the confidentiality protection techniques are highly demanded for the NVM-based NN computing systems.

2.3 Compensation for the NVM Vulnerabilities

Recall that the vulnerabilities of the NVMs mainly include the risk of data leakage and limited endurance. Previous studies have also tried to compensate for the vulnerabilities.

An active research area of NVM is dealing with the limited endurance problem. 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 [27]. 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, the work in [10] proposes a structured gradient sparsification (SGS) scheme which reduces write frequency, together with an aging-aware row swapping (ARS) method for the wear-leveling. 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 [11], [28]. So far, the limited endurance is not a major vulnerability that threatens the NVM-based system security.

There are also many attempts on protecting the lingering confidentiality in the NVMs. However, to our best knowledge, most of the prior approaches are proposed for the main memory applications, which are not adapt for the CIM architecture and NN application. Encryption techniques are widely applied to protect the confidential data remained in the NVM. Generally, an encryption solution ought to include three basic components, controlling how, when, and where to encrypt respectively. For the first component, typical approaches include advanced encryption standard (AES) [29] and counter mode encryption (CME) [30], which ensure that only the owners can access the plaintext data by keeping secret keys. For the last two components, as introduced before, bulk and incremental [7] encryption methods give the location and timing to perform the encryption. However, as the NN computation involves heavy weights, the above conventional encryption approaches will incur tremendous performance overhead. Therefore, the amount of encrypted weights should be substantially reduced. As shown in Fig. 2, 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.

3 ATTACK MODEL

3.1 Goal of Security Protection

Owing to the data persistence of NVM, an adversary 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 [7], [20]. Thus, the attack model in this work is confidentiality attack on the
NN weights. With the AI devices becoming increasingly ubiquitous and mobile, the attackers have many opportunities to obtain the physical access to the CIM hardware. Therefore, it is necessary to find solutions for protecting the confidential NN weights. The goal is to thwart the threat of leaking the deployed NN models at negligible overhead.

Algorithm 1. Exhaustive Analysis of Channel-Wise Encryption

**Input:** Validation dataset $\chi$
**Input:** Encryption function $Encrypt$
**Input:** Neural network weights $W$

1. $W_{back} \leftarrow W$
2. $L \leftarrow W.LayerNum$
3. For $l = 1, 2, \ldots, L$ do
4.   $C \leftarrow W(l).ChannelNum$
5.   For $c = 1, 2, \ldots, C$ do
6.     $W_{back} \leftarrow W_{back}$
7.     $W(l, c) = Encrypt(W(l, c))$
8.     Validate the encrypted model
9.   end for
10. end for

3.2 Principles of Security Solution

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

1) **Functionality** 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 starts from the front layers to the end, it is preferable to encrypt fewest weights in the front layers.

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

4) **No Vulnerability Window.** The solution shall keep the system secure at all time, i.e., at any moment, most of the weights should be encrypted and eliminate the attack window. 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 reveal the weights.

4 Motivational Examples

4.1 Where and How to Encrypt

Encrypting all weights can intuitively be more secure. However, as mentioned before, it is inefficient to encrypt all the weights due to the unacceptable overhead. Moreover, it also widens the vulnerability window because when the system is powered off, it will take much more time to re-encrypt all the weights. Therefore, we must identify the most significant weights and partially encrypt them to reduce overhead. A straightforward idea of identifying the significant weights is analyzing the sensitivity of each weight by exhaustive search. Sensitivity is defined as the impact on the accuracy. However, exhaustive search has the following drawbacks.

On one hand, the exhaustive search incurs a high complexity. Assuming that we divide the weights into $G$ groups and encrypt them independently to observe their sensitivities, the complexity of the analysis will approach $O(G) \cdot O(\text{Test})$, where $O(\text{Test})$ represents the complexity of a validation round, and will linear increase with the instance number in the test dataset. As the group number $G$ increases, the time required for the search will be linearly 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 coarser grouping case that each group contains 1,000 weights in VGG-16, without considering the overlapping, the analysis will consume $250 \times 138k = 34.5$ million seconds, approximately 399 days. Moreover, coarsely grouping the weights will introduce quantities of ineffective encryption on the insignificant weights. The situation will become more complicated if considering irregular and cross-layer grouping. Thus, such exhaustive approach is impractical for identifying significant weights.

On the other hand, the encryption effectiveness is not satisfactory. We make experiments of encrypting a ResNet-18 network [31] trained on the CIFAR-10 dataset, as shown in Fig. 3. The adopted encryption methods include $encrypt-at-0$, $1$, and $random$, which encrypt the target weights as 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 the computation. Therefore, we must find an efficient method to identify the significant weights.

4.2 When to Encrypt
Another critical problem is when to encrypt the weights. We consider a typical application scenario that is 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. 4, 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 adversary 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.

5 Methodology
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. 5, which contains two main 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.

5.1 SFGE: Sparse Fast Gradient Encryption

Inspiration. The fast gradient sign method (FGSM) [26] 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, then make small changes on the weights to cause rapid deterioration on the NN performance. The fast gradient method can help to find the most significant gradient descent direction.

Another interesting discovery is the sparse nature of the gradient with respect to the weights. Many studies have explored the gradient sparsification approaches [32] to mitigating the bandwidth requirements in distributed NN training system. For example, deep gradient compression (DGC) [32] demonstrates that one can only preserve approximately 0.1 percent of the gradients to achieve comparable accuracy with normal training. This enlightens us that the models can also be greatly impacted by sparse perturbations. Recall that the third principle is not incurring large overhead. It is necessary to apply a sparsification on the fast gradient.

Problem Formulation. Let $\Theta$ be the weight parameters of an NN model, $\Theta$ be the perturbation matrix (whose elements are also the encryption keys) added on the original weights, $\chi$ be the validation dataset, $x$ be the input to the models sampled from $\chi$, $y$ 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 number of encrypted weights should be within an acceptable boundary owing to the constrained encryption budget. Here we denote the encryption budget as $N$. We set a selection matrix $Mask$ which contains only 0 and 1, to generate the sparse encryption key matrix $\Theta$. Thus, the number of 1s in $Mask$ should be smaller than $N$.

Second, the encrypted weights should not significantly fall outside the normal distribution range, otherwise they will become outliers and easily detected by the adversaries. Conventional encryption solutions ensure that there is no method that can be more efficient than brute-force collision, e.g., the cracking complexity of 256-bit AES key approaches $2^{256}$. In our solution, as we only encrypt an extreme small proportion of the weights, the cracking complexity depends on the difficulty of finding where the encrypted weights are located. For example, when encrypting $N$ weights from $M$ weights, the complexity of brute-forcely finding the locations will be $C(M, N)$. Therefore, the encrypted weights should appear the same with other weights, or the adversaries are easier to crack the keys by distinguish the outliers. As we apply a perturbation on each encrypted weight, the values of encrypted weights $\Theta$ should be constrained, and we assume the maximum perturbation intensity as $\epsilon$. In summary, since our goal is to find the optimal $\Theta$ to degrade performance, the encryption problem be modeled as

$$\begin{aligned}
\max_{\Theta} & \sum_{(x,y) \in X} J(\Theta + \tilde{\Theta}, x, y) \\
\text{s.t.} & \\
& \tilde{\Theta} = \Theta^* \odot Mask \\
& 1(Mask) \leq N \\
& \max(|\tilde{\Theta}|) \leq \epsilon.
\end{aligned}$$

(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:

$$\Theta^* = \sum_{(x,y) \in X} \nabla_\Theta J(\Theta, x, y).$$

(2)

However, the fast gradient $\Theta^*$ is still dense. Since there is an encryption amount constraint, we 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 $\Theta$ shows 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-$N$. At final, to reduce the complexity of the keys, we only preserve the sign of corresponding fast gradient. Therefore, the fast gradients preserved becomes

$$Mask \leftarrow |\Theta^*| \geq thr,$$

(3)

$$\tilde{\Theta} = \epsilon \cdot \text{sign} \left( \left( \sum_{(x,y) \in X} \nabla_\Theta J(\Theta, x, y) \right) \odot Mask \right).$$

(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”. Algorithm 2 concludes the overall algorithm and process.

**Algorithm 2. SFGG: Sparse Fast Gradient Encryption**

| Input: Validation dataset $\chi$ and size $b$ |
| Input: Neural network weights $\Theta$ ($L$ layers) |
| Input: Cost function: $J(\Theta, x, y)$ |
| Input: Constraints: encryption amount per layer $N$, encryption intensity $\epsilon$ |
| Output: Encryption keys: $\tilde{\Theta}$ |

1: $\Theta^* \leftarrow 0$
2: for $(x, y)$ in enumerate($\chi$) do
3: $\Theta^* \leftarrow \Theta^* + \nabla_\Theta J(\Theta, x, y)$
4: end for
5: for $\Theta^*_i$ in enumerate($\Theta^*$) (i = 1, 2, ...,$L$) do
6: Select threshold: $thr_1 \leftarrow \text{top } N$ of $|\Theta^*_i|$
7: $Mask \leftarrow |\Theta^*_i| \geq thr_1$
8: $\tilde{\Theta}_i \leftarrow \epsilon \cdot \text{sign}(|\Theta^*_i| \odot Mask)$
9: end for
10: for $\tilde{\Theta}_i$ in enumerate($\Theta^*$) (i = 1, 2, ...,$L$) do
11: $\tilde{\Theta}_i = \tilde{\Theta}_i + \tilde{\Theta}_i$
12: end for

**5.2 i-SFGE: Incremental SFGE**

The SFGG scheme generates all the keys in one forward-backpropagation round. While the SFGE method is fast and efficient, the encryption effectiveness is not the most prominent because the model sensitivity may vary when subtle changes are applied on the weights. When modifying a weight, the sensitivities of the overall weights will probably be different from that of the vanilla weights. Motivated by this, we introduce an incremental SFGE, referred as i-SFGE, to generate a stronger group of keys that can more effectively disable the NN models by incrementally generating the keys one by one. The process goes as follows. The $n_{th}$ key for each layer will be generated based on the model that has been partially encrypted by the already generated keys.

$$\Theta_n = \Theta_{n-1} + \tilde{\Theta}_{n-1}.$$  

(5)

Then the historical gradients will be zeroed, and another round of gradient generation will be performed on current model. Note that each gradient generation round will produce only one key for each layer, thus the maximum gradient of every layer will be selected out.

$$\Theta^* = \sum_{(x,y) \in X} \nabla_\Theta J(\Theta_n, x, y)$$

(6)

$$Mask \leftarrow |\Theta^*| = \max(|\Theta^*|).$$

(7)
Finally, we take the sign of the selected gradient sign as the key, with multiplying the configured encryption intensity, following the same rule with SFGE. So far, we successfully get the \( n_{th} \) key.

\[
\Theta_n = \epsilon \cdot \text{sign}(\Theta^* \odot \text{Mask}).
\]

We conclude the process by pseudo-code as shown in Algorithm 3. Overall, the gradient generation process will be performed for \( N \) times to obtain the expected number of keys, thus increases the key generation complexity by \( N \) times. Meanwhile, a suppression mechanism is set to prevent the weights from being selected twice or more. i-SFGE greedily select the most sensitive points at the present model state, which is promising to eliminate ineffective encryption and achieve better encryption effectiveness.

**Algorithm 3. i-SFGE: Incremental Sparse Fast Gradient Encryption**

**Input:** Validation dataset \( \chi \) and size \( b \)

**Input:** Neural network weights \( \Theta \) (L layers)

**Input:** Cost function: \( f(\Theta, x, y) \)

**Input:** Constraints: encryption amount per layer \( N \), encryption intensity \( \epsilon \)

**Output:** Encryption Keys: \( K \)

1. for \( n \) in range(\( N \)) do
2. \( \Theta^* = 0 \)
3. for \( (x, y) \) in enumerate(\( \chi \)) do
4. \( \Theta^* \leftarrow \Theta^* + \nabla_{\Theta} f(\Theta, x, y) \)
5. end for
6. for \( \theta^* \) in enumerate(\( \Theta^* \)) (i = 1, 2, …, L) do
7. Select key : \( ind_{i} \leftarrow \text{max of } |\theta^*| \)
8. while \( ind_{i} \) in \( K \) do
9. \( \theta^*_i(ind_{i}) = 0 \)
10. end while
11. end while
12. Mask = Zeros(\( \theta^*_i \).shape)
13. Mask(\( ind_{i} \)) = 1
14. \( \Theta_i \leftarrow \epsilon \cdot \text{sign}(\theta^*_i \odot \text{Mask}) \)
15. \( K_i \cdot \text{append}(ind_{i}, \text{sign}(\theta^*_i(\text{ind}_{i}))) \)
16. end for
17. for \( \theta_i \) in enumerate(\( \Theta \)) (i = 1, 2, …, L) do
18. \( \Theta_i \leftarrow \Theta_i + \Theta_i \)
19. end for
20. end for

5.3 Details of the Encryption

**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 \( \lfloor \log_2(L) \rfloor + \lfloor \log_2(M) \rfloor + 1 \) bits to record the encryption location, 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 -).

**Decryption.** The only operation introduced by the decryption is the addition on the weights. Therefore, the decryption only needs to add a negative item of the sparse fast gradient perturbations 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.** The key generation is an one-shot process for each model, i.e., for a well-trained model to be deployed, the SFGE or i-SFGE process only needs to perform once to generate the keys. When the systems run, the keys will be strictly protected and utilized to encrypt or decrypt the weights. Two basic operations should be implemented: the forward-backward propagation to generate the gradients, and the gradient sorting to select the most significant weight. Therefore, the complexity will be \( O(ST) + O(M \log_2(N)) \), where \( S \) denotes the instance number in the sampled dataset, \( T \) denotes the time overhead of processing one instance, and \( N \) represents the key number for each layer. The former one refers to the complexity of one generating the gradients for the weights, and the latter one represents the complexity of the sorting operations to select the gradients with largest absolute values. As for i-SFGE, because the key generation process will be performed for \( N \) times, the complexity will increase to \( O(NST) + O(NM) \).

**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 \( \epsilon \) 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 be written on the corresponding weight location. While encrypting more weights certainly results in a higher security level. Concurrently, the encryption intensity \( \epsilon \) also affects the encryption effectiveness. Larger \( \epsilon \) 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 and become outliers. Therefore, \( \epsilon \) must be carefully designed based on the weight distribution.

5.4 Variants of the Encryption

The above encryption design describes general cases that consider floating-point models. Specifically, we can further make variants on the approach, consider the NN accelerator characteristics, and make the encryption more practical and efficient. For example, in quantized NN models, when a single NVM device represents a multi-bit value, the perturbation can also be quantized and discretized, turning the conductance from one level to another. The following we will discuss more general bit-wise encryption variants.

The most common-seen accelerators, including the NVM-based CIM designs [33], [34] or CMOS-based FPGA [35] and ASICs [36], [37], generally utilize fixed-point numbers with bit-width of 8 or less, to implement the computations. Meanwhile, to achieve higher reliability, many CIM architectures [38] propose to design accelerators based on single-bit NVM devices, rather than multi-bit devices. They distribute a single weight to multiple cells for the value representation. Therefore, we further reduce the granularity of the encryption and implement bit-wise operations.

Instead of adding the floating-point perturbations, we only perform the encryption on the first two bits of the selected weights. We assume that the binary numbers are encoded by 2’s complement, and the encryption transformations are shown in Table 1. Assuming the scale of the quantization as \( 2^a \), the encryption is actually equivalent to...
To avoid overflow, the encryption will ignore the case of “01” when the gradient sign is positive and the case of ‘10’ when the gradient sign is negative, and eliminate the encryption.

Adding a perturbation with intensity of $2^{-1}$. Such bit-wise encryption operations bring two benefits. First, the encryption only modifies two bits for each encrypted weight, thus decreases the write overhead because the operated cells will be substantially reduced. Second, the encryption will be intensity-flexible, because the distributed ranges of the quantized weights in different layers always reflect on the quantization scale. Therefore, as we always operate on the first two bits, the intensity will become adaptive. This enhances the adaptability, and effectiveness of encryption. On one hand, it fully utilizes the full distributed space to apply the largest possible perturbation. On the other hand, it also constrains the perturbation intensity and prevents the encrypted weights from jumping out the normal range and being easily detected.

Note that the encryption can be implemented by simple logic or look-up tables. Assuming the two bits as $B_0\cdot B_1$, the gradient sign bit as $P$ ($P = 1$ represents positive and $P = 0$ represents negative), the logic representation will be $\overline{B_0} = (B_0 + B_1)\overline{P} + B_0B_1$, $B_1 = \overline{B_1}$. Meanwhile, as shown in Table 1, when eliminating the cases “01” and “10” respectively, the transformations under positive sign is exactly the reverse of negative sign. Therefore, the decryption process will be the same as illustrated in Section 5.3. The decryption logic will be exactly the same as the encryption, just flipping the sign bit. Moreover, we follow the same process as SFGE or i-SFGE to generate the significant weight locations and their gradient signs. Therefore, the SFGE keys and corresponding key generation process and complexity remain unchanged.

5.5 RES: Runtime Encryption Scheduling

A runtime encryption scheduling is highly demanded to keep the CIM system secure all the time, simultaneously narrowing the vulnerability window when re-encrypting the weights. The conventional way, which decrypts before starting work and encrypts after ending the work, has a vulnerability window that the attackers still have opportunities to steal the model by interrupting the system during runtime or during the re-encryption window. While the NN computing always starts from the first layer, and end in the last layer. Dependencies exist between the layers, i.e., the input of a layer is the output of its previous layer. Commonly-seen 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, because the encryption can also be simply done layer-by-layer. The whole process is shown as Fig. 5, the layer will only be decrypted when the program comes, and be encrypted after the work has been done. Pre-decryption can also hide the decryption latency during runtime. At most running time, only one layer is in plaintext form. Only in the very short interleaved moments will the two adjacent layers be in plaintext form simultaneously.

Another commonly-seen method is cross-layer scheduling, which fully utilizes the parallelism across the layers [39]. Because sliding windows are used to convolve the feature maps, 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. While we only consider the layer-by-layer scheduling in the following experiments.

Discussion. RES will incur write operations for the encryption in every inference round. This raises concern about the lifetime of the systems. Because the SFGE keys are fixed, frequently operating on the same cells will certainly wear out them. While 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 is totally predictable.

Another concern about the security of RES is that the adversaries may still have opportunity to obtain the whole weights by disrupting the system multiple times. While the prerequisites are that: 1) the adversaries have prior knowledge of the layer scheduling and then they can interrupt the system exactly when the desired layer is running; and 2) once powered off, the weights of the running layer will remain in plaintext form. Therefore, three ways can compensate for the vulnerability. First, introduce obfuscation in the layer scheduling to hide the running information, as similarly discussed in [21]. Second, circuit-level innovations can avoid the adversary from correctly reading the data without authorization [13], while require additional circuit support. Third, set up emergency encryption mechanism with build-in temporary battery to tolerant the malicious interruption or unexpected power failure. Once detected as interrupted, the module can be activated and encrypt the running layer immediately. Because the encryption weight number of a single layer is around only 20 to 30, the emergency encryption can be finished in a very short time, and the equipped temporary battery capacity can be small.

6 EXPERIMENTS

6.1 Experiment Settings

We investigate the accuracy influence and protection effectiveness of our solution. The experiments are constructed on the ResNet [31] (with 18, 50, and 101 layers) and VGG-16 models [12] with the ImageNet dataset, and the SSD [40] model with the VOC dataset [41]. The evaluation metrics include: 1) the accuracy influence; 2) the security analysis; and 3) the overhead. There are two main parameters involved in the experiments: the encryption amount per layer $N$ and
When considering the or the intention approaches, continuing to enlarge will not be profitable because the performance has been represented by the number of . We also visualize the predictions of both under different intensity .

Fig. 7. The validation accuracy of the encrypted VGG-16 models versus .

Fig. 8. Contrasting the encryption effectiveness of SFGE and i-SFGE.

6.2 Accuracy Influence of Encryption

Encrypting the NN models by SFGE has a significant impact on the accuracy performance. Recall that our goal is to destroy the prediction ability of the NN models, more influence on the accuracy indicates more effective encryption.

Classification Models. Fig. 6 shows the validation accuracy of the NN models versus the encryption amount per layer . Four interesting conclusions can be figured out from the results. First, the accuracy influence increases with . Because with encryption amount increases, the encrypted weights will increasingly vary from the original weights, and more computational errors will be introduced. Second, there exists a turning point on the curves. For example, in the curve of ResNet-101, when approaches 15, continuing to enlarge will not be profitable because the performance has been already fully deteriorated. Third, the trend of accuracy drop is closely related to the NN depth. See the results of ResNet-18, 50, and 101, the ResNet-101 curve 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 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× heavier weights of VGG-16 than ResNet-18. In this scenario, increasing or the intensity can improve the encryption effects.

Fig. 7 shows the validation accuracy of encrypted ResNet-18 versus the encryption intensity . The performance of encrypted models with larger degrades more quickly than the ones with smaller ones. However, there still exist an intensity limit to enhance the encryption concealment, as will be discussed below.

Contrasting SFGE and i-SFGE. We compare the encryption effectiveness of SFGE and i-SFGE methods as introduced in Section. Fig. 8 shows the accuracy when encrypting the VGG-16 network by the two key generation methods. As can be seen, i-SFGE demonstrates much better performance than SFGE. Under the same encryption budget, the keys generated by i-SFGE can degrade the accuracy more significantly, i.e., can better identify the significant weights.

Object Detection Model. We also evaluate the effectiveness of our solution on object detection models. We select a single-shot multibox detector (SSD) [40], to test on the widely-used Pascal VOC dataset [41] to demonstrate the encryption effect. As shown in Table 2, with an encryption amount of 30 or 40, and an intensity of 0.1 or 0.2, the evaluated mAP demonstrates a significant drop. Therefore, our solution also works for the object detection applications.

Visualization. We also visualize the predictions of both classification and object detection models under plaintext form and ciphertext form. As can be seen in Fig. 9, the encryption will completely disable the NN models. The predictions are absolutely incorrect and cannot provide any useful information. Therefore, even if the encrypted models are leaked, the attackers are still unable to make use of them.

Evaluating Bit-Wise Encryption. When considering the fixed-point weights that are frequently utilized in NN accelerators, we apply bit-wise encryption to protect the deployed models. As shown in Table 3, two conclusions can be figured out. First, under the same encryption number, bit-wise
encryption provides an adaptive intensity, thus fully utilizes the dynamic range and degrades the accuracy more. Second, the perturbation ranges are limited, which ensures that the encrypted weights will not fall outside the normal range. Most importantly, the bit-wise encryption can greatly reduce the write complexity and overhead, especially in the NVM architectures by mapping a single weight with multiple cells. By encrypting less than 0.0001 percent of the overall bits, the accuracy can drop to near zero. Moreover, it is reasonable that the conclusions mentioned in floating-point cases still established under bit-wise encryption because both the essence is disturbing the key weights.

**Effectiveness Under Variation.** As multi-level NVM cells usually suffer from process variation, the mapped values will deviate from expected. We demonstrate that the encryption solution is still effective when meeting the variation. We consider a generic variation model [42], i.e., a cell has multiple conductance levels and the variations satisfy Gaussian distribution with zero mean and specific standard deviations. As can be seen in Fig. 10, the accuracy becomes increasingly unstable when the standard deviation of the variation increases. While we can figure out that the encryption is still effective on the deployed models under variation. The underlying reason is that although the process variation may change the deployed weight values, the significance property of the weights still remain.

### 6.3 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, we consider from various aspects as following.

<table>
<thead>
<tr>
<th>Model</th>
<th>N/C3</th>
<th>Backbone</th>
<th>Head</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD</td>
<td>30</td>
<td>0.1</td>
<td>✓</td>
<td>77.43</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.025</td>
<td>✓</td>
<td>51.07</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.2</td>
<td>✓</td>
<td>12.25</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.05</td>
<td>✓</td>
<td>10.02</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>0.1</td>
<td>✓</td>
<td>49.13</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>0.025</td>
<td>✓</td>
<td>36.15</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>0.2</td>
<td>✓</td>
<td>5.47</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>0.05</td>
<td>✓</td>
<td>3.15</td>
</tr>
</tbody>
</table>

⁴We partition the network into two parts: “backbone” for the main part that produces the features, and “head” for the part that implements the predictions. We find that the dynamic range of head layers is smaller than the backbone, thus we decay the encryption intensity of the head, $\epsilon_h$ represents the intensity for encrypting backbone, and $\epsilon_b$ for encrypting head. We evaluate the mean average precision (mAP) on VOC2007 test set [41].

<table>
<thead>
<tr>
<th>Model</th>
<th>N/C3</th>
<th>Param. Bits</th>
<th>Encrypt Bits</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD</td>
<td>30</td>
<td>11.7 MB</td>
<td>0</td>
<td>67.93</td>
<td>87.91</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>11.7 MB</td>
<td>48b</td>
<td>51.07</td>
<td>51.07</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>11.7 MB</td>
<td>96b</td>
<td>12.25</td>
<td>12.25</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>11.7 MB</td>
<td>144b</td>
<td>10.02</td>
<td>10.02</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>11.7 MB</td>
<td>192b</td>
<td>36.15</td>
<td>36.15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>N/C3</th>
<th>Param. Bits</th>
<th>Encrypt Bits</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD</td>
<td>30</td>
<td>138 MB</td>
<td>0</td>
<td>71.35</td>
<td>90.15</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>138 MB</td>
<td>0.16Kb</td>
<td>66.40</td>
<td>86.93</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>138 MB</td>
<td>0.32Kb</td>
<td>52.59</td>
<td>76.31</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>138 MB</td>
<td>0.48Kb</td>
<td>24.49</td>
<td>45.68</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>138 MB</td>
<td>0.64Kb</td>
<td>6.268</td>
<td>16.11</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>138 MB</td>
<td>0.96Kb</td>
<td>1.570</td>
<td>4.952</td>
</tr>
</tbody>
</table>

⁵The weights of ResNet-18 and VGG-16 networks are both uniformly quantized to 8-bit numbers.
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 crack the encryption with a much lower 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. 11, 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 $\epsilon$ brings better encryption effect, it also increases the risk of being detected. As shown in Fig. 11, when $\epsilon$ reaches 0.5, many encrypted weights will jump out the normal range and become outliers. Thus, when $\epsilon$ lies in a reasonable range, the adversaries will have to brute-force search the encrypted weights. The complexity far exceeds $2^{256}$ (the complexity of AES with 256-bit key), e.g., recovering 20 weights from $3 \times 3 \times 64 \times 64$ weights in a convolution layer takes approximately $8 \times 10^{72}$ tries.

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. 12, there are extremely small fluctuations on the mean $\mu$ and variance $\sigma$. Moreover, we calculate the norm squared difference of the original histogram and the encrypted one, which reaches only $5.43 \times 10^{-5}$. Therefore, the adversary 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 the 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 $\binom{1000}{20} \approx 3.4 \times 10^{41}$. This concern is raised by that the encrypted models may still maintain the same sensitivity as the plaintext models, so the attackers may collide with the encrypted weights by performing the gradient generation again. The mean recall represents the mean value of the recall rates of all layers. As shown in Table 4, the recall of ResNet models are low. Although the VGG-16 shows much larger recall, it is still 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 4, 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 defended under SFGE.

Security of Runtime Encryption Scheduling. To ensure the computational correctness, the weights must be decrypted back to plaintext form when performing the corresponding computations. Owing the the proposed RES method, at most time will one layer of weights be decrypted. Therefore, even the attackers disrupt the system when working, they only obtain a model with only one layer decrypted. We evaluate the accuracy when decrypting one of the layers in Fig. 13. As can be seen, the accuracy still remain at a extremely low level when the model is with one layer decrypted. However, as the depth and sensitivity of the layers vary, there will be slight differences in the accuracy when decrypting different layers.

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th>Classification Accuracy</th>
<th>Encryption Config</th>
<th>FGSM Adv. (top-1) (Intensity: 0.05)</th>
<th>Mean Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Encrypted</td>
<td>Baseline</td>
<td>Encrypted</td>
<td>Plaintext</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>ImageNet</td>
<td>Top-1: 70.3% (-69.05%)</td>
<td>Top-5: 2.45%</td>
<td>69.75%</td>
<td>89.07%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Baseline Top-1: 76.13%</td>
<td>Baseline Top-5: 91.32%</td>
<td>76.13%</td>
<td>92.86%</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>ImageNet</td>
<td>Top-1: 0.33% (-75.69%)</td>
<td>Baseline Top-5: 77.38%</td>
<td>77.38%</td>
<td>93.54%</td>
</tr>
<tr>
<td>VGG-16</td>
<td>ImageNet</td>
<td>Top-1: 0.81% (-70.77%)</td>
<td>Encrypted Top-5: 3.47%</td>
<td>71.59%</td>
<td>90.38%</td>
</tr>
</tbody>
</table>

Pre-Deployment: Key Generation Complexity. To distinguish the significance of all the weights, gradients will be utilized to obtain the keys. Therefore, to generate the gradients, we need to sample a subset of data from the training or validation dataset to execute the forward and backward propagations. There exist a trade-off between the sampled dataset size and the encryption effectiveness. On one hand, as each data instance will be processed, the complexity of gradient generation linearly increases with the amount of sampled data. On the other hand, to obtain more generalized gradients to distinguish the important weights, the sampled dataset should have a good generalization, which requires a sufficient amount of data instances. As shown in Fig. 14, when increasing the data amount, the encryption effectiveness becomes better as the accuracy drops more stably, while accompanied with a longer key generation time. According to the experimental observations, 10,000 samples are adequate to distinguish generalized gradients for ImageNet dataset. We test the generation time of various models based on an NVIDIA GeForce RTX 2080 Ti platform. As Table 5 shows, generating the keys for one model consumes from 22.5 seconds to 77.8 seconds when using SFGE scheme. i-SFGE will consume much more time because the keys are generated one-by-one, but still within 1 hour.

Runtime: Encryption 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

![Fig. 13. The accuracy of encrypted ResNet-18 when decrypting one of the layers. The “baseline” represents the accuracy of the fully encrypted model. The tag of x-axis represents the name of the decrypted layer and the bar shows the accuracy.](image1)

![Fig. 14. The average validation accuracy versus the number of samples when generating the keys for floating-point ResNet-18 model. The encryption configurations are $N=20$, $\epsilon=0.2$. Each point is averaged based on the results of 10 independent experiments. The data are randomly sampled from the ImageNet train dataset which includes 1.28 million images in total.](image2)
TABLE 5
Encryption Time Overhead Evaluation

<table>
<thead>
<tr>
<th>Model</th>
<th>Key Num.</th>
<th>Generation Time</th>
<th>Runtime Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SFGE</td>
<td>i-SFGE</td>
</tr>
<tr>
<td>ResNet-18</td>
<td>20</td>
<td>22.5s</td>
<td>434.8s</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>30</td>
<td>50.6s</td>
<td>1447.9s</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>20</td>
<td>77.8s</td>
<td>1534.9s</td>
</tr>
<tr>
<td>VGG-16</td>
<td>30</td>
<td>73.6s</td>
<td>2246.8s</td>
</tr>
</tbody>
</table>

presented in NVSim [43], the write latency of PCM and RRAM achieves 416.2 ns and 100.6 ns respectively. The decryption only incurs one write on each encrypted weight, therefore the overall decryption latency will be expected as 416.2 N ns (PCM-based) and 100.6 N ns (RRAM-based) respectively. As shown in Table 5, taking the ResNet-101 as an example, the encryption amount of the first layer is \( \min(7 \times 7 \times 3 \times 64 \times 0.1\% , 20) = 10 \). Thus, the latency will be 4.16 µs and 1.01 µs respectively. Such latency overhead is negligible in most applications, e.g., the video frame rate usually ranges from 30 to 200 frames per second, consuming 5 ms per frame at least. Compared to 5 ms/frame, the encryption latency (≤ 5 µs) only takes less than 0.1 percent of the time.

7 FURTHER DISCUSSION

While we have discussed and demonstrated the advantages of our solution and the application on NVM-based in-memory NN computing systems, we still need to focus on the potential application on the CMOS-based accelerators and the drawback of our solution.

The Drawback: Good Weight Initialization. Although we have demonstrate the effectiveness and security of the proposed solution, there are also drawbacks that should be considered. Because only a extremely small proportion of weights are modified, the structure and distribution of the weights are preserved, thus the encrypted models still provide good weight initialization. As shown in Fig. 15, when the models are initialized with the encrypted weights, the losses will drop much faster than random initialization. This means that the attackers can still obtain some information from the adversarial attacks, as the encrypted models can help to accelerate and improve the convergence of training similar or same tasks. Our future work will further improve the encryption effectiveness and make the encrypted models closely equivalent to random-initialized models.

Potential Application on CMOS Accelerators. The main motivation of our work starts from the vulnerability of NVM systems rendered by the non-volatility. However, the volatile memory-based accelerators may also be threatened by adversaries. Taking FPGA as an example, when powered down, the configuration file (bitstream), weight file, and other necessary files are stored in non-volatile memory (e.g., disk or Flash). When booting the system, the bitstream will be loaded to the FPGA chip for configuration, and the other files will be loaded into a faster, volatile memory (usually DRAM) to accommodate the intermediate data and weights during computing because the on-chip memory capacity is far insufficient. However, we generally regard that only the FPGA chips are trusted, and both the peripheral Flash or DDR are untrusted. Current security approaches protect the designs and models by applying encryption on the bitstream [44] and model files in Flash, while the weights remain in plaintext form in DRAM, which is still vulnerable to memory attacks such as cold boot attack [25]. To ensure a higher security level, the data in DDR should also be encrypted. While the bitstream decryption is one-shot, the weight decryption will be done repeatedly as the FPGA cores only fetch a small proportion of them each time. This brings a challenge that the encryption will incur considerable latency. Therefore, reducing the encryption amount is also beneficial for reducing the encryption/decryption overhead, and our solution can also be transferred for protecting the model security in such accelerators.

8 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. Further improvements have also been made to enhance the encryption strength and efficiency by incrementally generating the keys or performing bit-wise encryption. Experimental results have demonstrated the effectiveness and robustness of our solution.

ACKNOWLEDGMENTS

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB0105000, and Grant 2017YFA0207600, in part by National Natural Science Foundation of China under Grant U19B2019, Grant 61832007, Grant 61621091, and Grant 61720106013, in part by Beijing National Research Center for Information Science and Technology (BNRist), and in part by Beijing Innovation Center for Future Chips. The
work of Xiaoming Chen was supported by Beijing Academy of Artificial Intelligence (BAAI).

**References**


---

Yi Cai received the BS degree in electronic engineering from Tsinghua University, Beijing, China, in 2017. He is currently working toward the PhD degree with the Department of Electronic Engineering, Tsinghua University, Beijing, China. His research mainly focuses on deep learning acceleration and emerging non-volatile memory technology.
Xiaoming Chen (Member, IEEE) received the BS and PhD degrees in electronic engineering from Tsinghua University, Beijing, China, in 2009 and 2014, respectively. He is now an associate professor with the Institute of Computing Technology, Chinese Academy of Sciences, China. His current research interests include electronic design automation and computer architecture design for processing-in-memory systems. He served on the Organization Committee of Asia and South Pacific Design Automation Conference (ASP-DAC) 2020 and also served on the Technical Program Committees (TPCs) of Design Automation Conference 2020, International Conference On Computer Aided Design (ICCAD) 2019, ASP-DAC 2019, International Conference on VLSI Design 2019 & 2020, Asian Hardware Oriented Security and Trust Symposium (AsianHOST) 2018 & 2019, and IEEE Computer Society Annual Symposium on VLSI (ISVLSI) 2018. He was a recipient of the 2015 European Design and Automation Association (EDAA) Outstanding Dissertation Award and the 2018 DAMO Academy Young Fellow Award.

Lu Tian received the BS and PhD degrees in electronic engineering from Tsinghua University, China, in 2011 and 2017, respectively. From 2017 to 2019, she was a post-doc with NICS Lab at Tsinghua University, China. Since 2019, she has been a researcher at AI group, Xilinx Inc., China. Her current research interests include pattern recognition and machine learning.

Yu Wang (Senior Member, IEEE) received the BS degree and PhD degree (with honor) from Tsinghua University, Beijing, China, in 2002 and 2007, respectively. He is currently a tenured professor and chair with the Department of Electronic Engineering, Tsinghua University, China. His research interests include application-specific hardware computing, parallel circuit analysis, and power/reliability aware system design methodology. He has authored and coauthored more than 250 papers in refereed journals and conferences. He has received Best Paper Award in ASPDAC 2019, FPGA 2017, NVMSA17, ISVLSI 2012, and Best Poster Award in HEART 2012 with nine Best Paper Nominations. He is a recipient of DAC Under-40 Innovator Award, in 2018. He served as TPC chair for ICFPT 2019, ISVLSI 2018, ICFPT 2011 and finance chair of ISLPED 2012–2016, and served as the program committee member for leading conferences in EDA/FPGA area. Currently he is serving as an associate editor for the IEEE Trans on Circuits and Systems for Video Technology, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, and ACM Transactions on Embedded Computing Systems. He is an ACM senior member. He is the co-founder of Deephi Tech (acquired by Xilinx in 2018), which is a leading deep learning computing platform provider.

Huazhong Yang (Fellow, IEEE) received the BS degree in microelectronics, in 1989, the MS and PhD degrees in electronic engineering, in 1993 and 1998, respectively, all from Tsinghua University, Beijing, China. In 1993, he joined the Department of Electronic Engineering, Tsinghua University, Beijing, China, where he has been a professor since 1998. He was awarded the Distinguished Young Researcher by NSFC, in 2000, Cheung Kong Scholar by the Chinese Ministry of Education (CME), in 2012, Science and Technology Award First prize by China Highway and Transportation Society, in 2016, and Technological Invention Award first prize by CME, in 2019. He has been in-charge of several projects, including projects sponsored by the national science and technology major project, 863 program, NSFC, and several international research projects. He has authored and co-authored more than 500 technical papers, 7 books, and more than 180 granted Chinese patents. His current research interests include wireless sensor networks, data converters, energy-harvesting circuits, nonvolatile processors, and brain inspired computing. He has also served as the chair of Northern China ACM SIGDA Chapter science 2014, general co-chair of ASPDAC’20, navigating committee member of AsianHOST’18, and TPC member for ASPDAC’05, APCCAS’06, ICCCAS’07, ASQED’09, and ICGCS’10.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.