Generating Robust DNN with Resistance to Bit-Flip based Adversarial Weight Attack

Liang Liu, Yanan Guo, Yueqiang Cheng,Youtao Zhang, Jun Yang

Abstract—Rowhammer Attack, a new DRAM-based attack, was developed exploiting weak cells to alter their content. Such attacks can be launched at the user level without requiring access permission to the victim memory cells. Leveraging such attacks, a new bit-flip-based adversarial weights attack (BFA) was developed targeting deep neural network models. When BFA attackers acquire a DNN model, they manipulate the existing DNN adversarial attack into locating vulnerable bits in the target DNN model. By flipping a subset of them using Rowhammer, they can crash that model within 30 trails. In this paper, we propose a lightweight and easy-to-deploy defense mechanism in the bit-level, Randomized Rotated and Nonlinear Encoding (RREC), which generates both robustness and fault-tolerant against BFA. Since flipping the most significant bit (MSB) in quantized data is too dangerous, we introduce randomized Rotation to obfuscate the bit order of model data and efficiently hide truly vulnerable bits with less vulnerable ones.Further, RREC reduces the average bit-flipped distance by more than 3x from the nonlinear encoding. It decreases the bit-flip distance among the majority of bits (including those vulnerable bits). Theoretically, RREC minimized the impact of a single bit BFA to 1/24 compared with baseline. Experimentally, RREC tolerates more than 17x flipped bits vs. baseline model and 4.8x and 5.7x more bits compared with the existing BFA defenses (4B QAT and WR) with 0.01x to 0.08x of runtime latency. Moreover, we evaluate RREC against a newly emerged attack, Targeted-BFA, and it improves the defense rate from 5% to 95%.

Index Terms—Machine Learning, RowHammer, Bit-Flip Attack

1 Introduction

As the demand of AI grows, more enterprises start to develop their proprietary of AI models. Although the capabilities of AI bring tremendous convenience to society, their intensive computational demands especially in model training are also limited by computing resources. As a result, Machine learning as a service (MLaaS) gradually becomes a popular practice, where users can leverage cloud-based machine learning tools rather than developing their in-house hardware or software infrastructure. As the computational or memory resource of the online server is expensive, using high-precision weights in DNN inference consumes a large amount of memory bandwidth. Many studies [1] have shown that using high-precision weights for inference unnecessarily lowers the inference efficiency, and proposed that using a compacted format such as 8-bit quantization could achieve the same accuracy as the full precision version. Users should also be cautious about the potential security threats, because the more compacted of data is encoded, the higher of risk of important bit being fault injected. Further, cloud servers typically share hardware devices among multiple users to maximize resource utilization. However, such sharing is still subject to security attacks that cannot be handled by existing security protection [2]. In this paper, we will discuss one such hardware threat from DRAM sharing.

Rowhammer [3] exploits vulnerabilities in DRAM cells and causes leaking or changing of the contents in nearby memory rows, where those rows are not addressed in the original memory access. Such attacks is hidden from the operating system and bypass permission checks at various levels. By manipulating Rowhammer, researchers developed a Bit-Flip based version of adversarial attack(BFA) targeting the DNN model, where adversarial attack [4] has been a secure hazard fulfilling the whole age of AI. BFA attackers use the algorithm of adversarial attack to compute the location of vulnerable bits and use rowhammer to flip them. Previous study [5], [6] showed that such bit-wise adversarial attack on a naïve DNN model can be as harmful as the traditional adversarial attack, where the result shows that flipping around 20 bits can cause a catastrophe to the entire model.

Three main categories of techniques can mitigate BFA, but each of them still has some adverse effects. Firstly, some studies tried to solve the Rowhammer problem at the hardware level [7], [8]. Researchers systemically implement the timer [9], [10], performance counters [11]–[13] or memory scanners [11] to monitor the system behaviors and cease the application when abnormal events are detected. However, the above defenses require specific hardware upgrading in either the DRAM or memory controller in the CPU, which poses challenges to deployed infrastructure.

Secondly, there also exist some the data-level defenses such as error detection and correction. However, the integrity check, e.g., checksum or Merkle-Tree [14], can only capture the errors but cannot correct them. Even a small error is detected, it deletes the entire data block and reloads them from the disk. Attackers can use Rowhammer to continually trigger the error and forces the user to spend tremendous time in I/O, which is equivalent to a DDoS attack [15] that can crash the server. On the other hand, Error Correction Code (ECC) introduces redundant bits to generate the data fault-tolerant, e.g., the Chipkill-Correct ECC [16]. However, generating multiple-bit correction is expensive. The previous study [17] shows that Rowhammer can attack more than one bit in a row

- *Liang Liu, Yanan Guo, and Jun yang are with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA, 15213. E-mail: {lil125,yag45, juy9}@pitt.edu*
- *Youtao Zhang is with the Department of Computer science, University of Pittsburgh, Pittsburgh, PA, 15213. E-mail: zhangyt@cs.pitt.edu*
- *Yueqiang Cheng is with Security Research Department at NIO Company.E-mail: strongerwill@gmail.com*

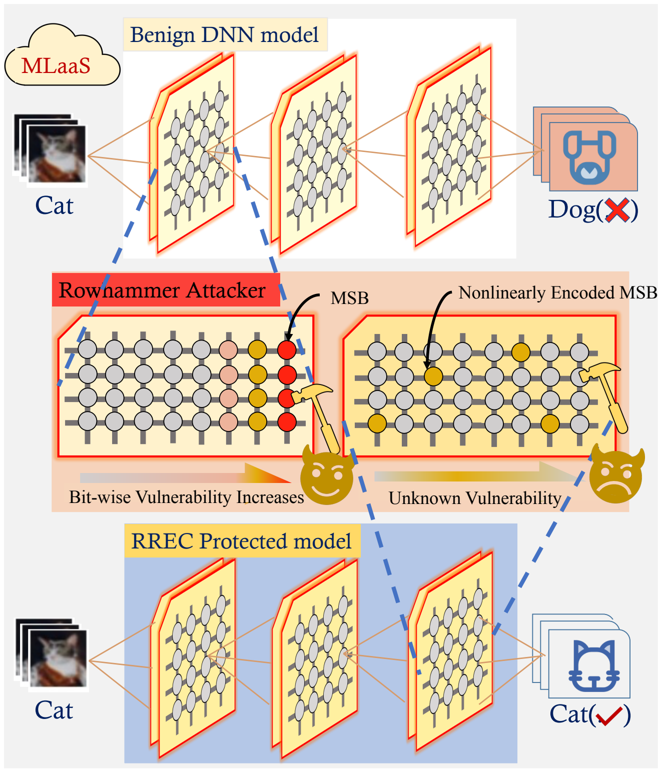


Fig. 1. **RREC Protects MLaaS Model in Bit-Level.** The benign model is easily combated by flipping the MSB, since they are always located at the left-most. RREC protects the model and rotates data in bits level. By applying the randomization in bits, attackers are impossible to locate the MSB. Further, RREC integrates nonlinear encoding to minimize the impact of single bit being flipped. Even if a MSB is accidentally flipped, its impact is still tolerable.

and bypass ECC, and more studies [8] show that Rowhammer is able to compromise Chipkill ECC.

Thirdly, there are works proposed to generate the reliability of the model. For example, Soft Error Resilience [18] triplicates the model weights into three copies and corrects the random bit flip by the majority vote; AEP [19] records the location of error bits and uses the mask to ignore those bits. However, improving the reliability only works for defending the random errors, but it cannot defend the white-box BFA since attackers also attack the most vulnerable bits. We also investigate the specific defenses against BFA to the model such as Binarization-Aware-Training model [20] or Weight Reconstruction [21]. They provide preliminary solutions to mitigate BFA, but the protection is still limited. QAT achieves promising robustness when using 1 bit binary weights (1B QAT), but the binary coding compromises too much accuracy (dropping from 92% to 86% for ResNet). If choosing 4-bit quantization mode (4B QAT), attackers can flip more bits (over 100 bits), and their model will still fail. Our Proposed RREC can achieve the same robustness as 1B QAT without compromising any accuracy.

Our proposal is based on the observation that Rowhammer alters the charge in a cell will increase the difficulty of altering other cells in the same row. BFA attackers tend to target the higher significant bits to flip because those bits contribute more difference to the benign data. Some existing obfuscation techniques, e.g., memory encryption (AES counter mode), encrypts the counter into a padding block and XORs the block padding with the plain text to get cipher text. It does not change the location of the MSB, so Attackers can still locate the MSB to flip. In this paper, we propose two bit-level defenses that hide the vulnerability at the bit level and spread the risk of higher significant bits into less significant ones.

Our first approach is inspired by the randomization method, e.g., Address space layout randomization (ASLR) [22]. It shuffles the memory address into a random order, preventing memory corruption exploitation. Unlike ASLR applying the randomization at the page level, we protect data at the bit level. In this paper, we found the lightest implementation of randomization, a randomized rotation, that hides the order of bits and turns a potential white-box BFA into a black-box attack, where the effectiveness would drop to 1/8.

Further, we integrate another bit-level defense above the randomized model to minimize the flipped bits' average distance. Before we started our implementation, we studied the robustness of existing quantization methods in defending BFA, such as clipping/pruning [23], nonlinear quantization [24], and dynamic quantization [25]. We observe that the nonlinear quantization method can reduce the distance of bits being flipped, and the experiments show an outstanding performance in defending BFA. Therefore, we enhance the inherent attribute of nonlinear quantization and develop a post-quantization-training mechanism to solve the optimal setting of nonlinear function to defend BFA. We achieve 3x of robustness gain through the optimized nonlinear encoding.

In summary, we generate robustness and fault-tolerance in the model, where although a proportion of model weights are comprised, the model still performs functionally. Experiments show that in defending the white-box attack, even if 100 bits were compromised, the accuracy is still above 70%, and in defending the black-box attack, the accuracy is still above 80% after 1000 bits are flipped. Compared with the previous BFA defense, RREC achieves 4.8x more robust VS. the 4B QAT and 5.7x more robust VS. weight reconstruction. We achieve the same robustness as 1B QAT. However, instead of 1B QAT suffering from an accuracy loss (6% to 7%), RREC enhances the accuracy of the original model, i.e., the accuracy of the VGG model grows from 87.7% to 88.8%, which outperforms the 1B QAT model vastly. Further, we evaluate the overhead of this defense, where the encoding only takes small proportions of computational overhead compared with the model inference overhead (from 0.01x to 0.08x). Moreover, RREC protects the model only at the bit-level, and it is orthogonal to the aforementioned model-level defenses such as AT or RS. They can work collectively to achieve better robustness. We can achieve even better performance by applying the QAT method.

2 Background

2.1 Rowhammer Attack

Modern DRAM-based memory chips consist of a two-dimensional array of cells. Each cell stores 1-bit information, represented by the charge of the capacitor in the cell. Though simple, such a structure was found to be vulnerable to disturbance errors induced by coupling [3]. That coupling effect activates a row in a DRAM bank and causes a little disturbance to its neighboring rows. With frequent activation/access to the same row (a row commonly stores 8k data), the disturbance on the neighboring rows will accumulate and eventually become significant enough to flip the stored bits before the rows get refreshed. With Rowhammer, attackers are able to change the memory data of a co-located application without even accessing it directly.

Rowhammer has already been successfully utilized to demonstrate many attacks [26], [27]. It has been proven that Rowhammer attacks can work on both DDR3 and DDR4 memories, even with Error-correcting codes (ECC memory) [28]. In [29], the authors show that Rowhammer attacks can be used to modify the weights of a functional DNN and make it generate random outputs.

2.2 Model Quantization

The model quantization [30] is to substitute model weights or intermediate results in memory from 32-bit float point into lower bits (e.g., 8 bits) of unsigned integer, which can reduce the overhead of intensive memory fetching. Previous studies show that 8-bit quantized models achieve nearly the same accuracy as the full-precision ones, with specific training strategies such as Quantization-Aware-Training(QAT) and Post-Training-Quantization (PTQ) [31].

Moreover, the floating point is highly prone to the bit-flip attack since the standard float point consists of an 8-bit exponent. For example, once the highest exponent bit is flipped from 0 to 1, the data will be amplified by 2^{256} times, which is a catastrophe to the model. The experiment shows that even 1 bit of flipping the exponent can crash a DNN model, so float point is not a preferred option if the threat model is prone to the BFA.

According to the different encoding methods, we can catechize the quantization into linear quantization and non-linear quantization. Let us use W to denote the full-precision float-point weights and B to denote the quantized weights. The linear quantization function $\mathcal{Q}_l$ is written as:

$$\mathcal{Q}_l(W) = \frac{\lfloor W \rceil}{\Delta w} \ \ , \ \ \mathcal{Q}_l^{-1}(B) = \Delta w \cdot B$$

where $\lfloor \cdot \rceil$ is a round operation, and it drops the precision; Δw denotes the quantization scale (also noted as quantization precision) of the corresponding layer, which is calculated as $\Delta w = \max(\mathbf{W}^l)/128$. Further, we can also express the integer weights via its bit-wise representation $B = \{b_1, b_2, \cdots, b_7\}$, to formalize quantized weights. The widest used quantization function is the signed linear encoding:

$$B = -128 \cdot b_7 + \sum_{n=0}^{6} 2^n b_n$$

Besides the linear quantization, recent studies explore more attempts on modifying the quantization function to improve precision or performance: 1. some works [23] apply the pruning and clipping to narrow down the quantization range, i.e. decrease the δw. it use the same quantization level to represent a smaller domain, which equally increase the precision; 2. more works dynamically manage the quantization length [25], i.e. adjust the quantization function $\mathcal{Q}(\cdot)$ and δw during training or inferring. It reduces the quantization bits for nonessential layer; 3, other works analysis the data distribution, and abandon the linear quantization function and apply nonlinear quantizer [24] to assign more precision to the majority of weights. For example, in this paper, we integrate a power function into our encoding:

$$\mathcal{Q}_{\text{nl}}^{-1}(B) = \text{sign}(B) \cdot \Delta w \cdot ((|B| + \alpha)^\gamma - \alpha^\gamma)$$

,where we choose α as the integer power term. According to the property of power function, the growing speed will be relative slow when the base B in is small. Empirically, the weights in DNN model are densely distributed at the close-to-zero zone, so the power function brings more precision to the majority of weights. In the design section, we will elaborate the detail of the implementation and its robustness gain.

2.3 Bit-Flip-Based Adversarial Weights Attacks

Adversarial attacks have become one of the main challenges in DNN security [32], [33]. Due to the nature of DNN, even small changes in inputs or weights could lead to huge differences in inference accuracy. General adversarial weight attacks maximize the effect of the attack by locating the weights with the highest gradients w.r.t the inference loss ℓ. After applying a small shift in such identified weights, there will be a significant change in the loss. Projected Gradient Descend(PGD) and Fast Gradient Sign Method(FGSM) [34] are commonly use by attackers.

Bit-Flip Attack: Bit Flip Attack (BFA) [6], [35] is a variant of adversarial weights attacks. This attack performs weight modification by flipping the bits of quantized DNN weights stored in the memory, utilizing well-developed Rowhammer tools. The main goal of BFA is to flip the optimal combination of bits in weights of a DNN model to maximize its inference loss.

As the majority of MLaaS are tightly restricted by the computation resource, model quantization compresss a DNN model and representing the model with lower memory bandwidth. BFA algorithm is based on the logic behind quantization. BFA shares a similar idea as the adversarial weight attack: to maximally increase the loss with a small shift in weights. The difference is that BFA shifts the weights by flipping their bits. The loss function is written as follow:

$$\begin{gathered}\mathcal{L}_1 = \ell(f(\mathbf{x}, \mathbf{B}'), \mathbf{y}) - \ell\left(f(\mathbf{x}, \mathbf{B}), \mathbf{y}\right) \\ s.t. \ \ \mathcal{H}(\mathbf{B}' - \mathbf{B}) < N_\epsilon\end{gathered}$$

where $N_\epsilon \in \mathbb{N}$ is a constant of Hamming distance that is limited for bit-flipping by such attack. Attackers solve this loss function and locate the bits with maximum gradient to flip. The goal is to increase the model's overall loss and finally crash the model.

Targeted Bit Flip Attack [6]: To attack only a few classes of model $\{\mathbf{x}_s, \mathbf{y}_s\}$, and fool the model to mis-classify it to $\{\mathbf{x}_t, \mathbf{y}_t\}$. It will not change the overall accuracy. The loss function can be written as follow:

$$\begin{aligned}\mathcal{L}_2 = \max(f(\mathbf{x}_s, \mathbf{B}'_s)) - \max(f(\mathbf{x}_s, \mathbf{B}'_t)) \\ + \lambda \cdot \ell(f(\mathbf{x}_n, \mathbf{B}'_\mathbf{n}), \mathbf{y}_n))\end{aligned}$$

where the $\max(f(\cdot))$ outputs the class with the highest confidence; $\mathbf{B}'_s$ and $\mathbf{B}'_t$ are the weights of the last-layer linking to the source and the target class; $x_n = \{x \setminus x_s, x_t\}$ are the class other than the source and target. The first term tend to enhance the target class and vanish the source class on all input x. The second term is to train the rest classes of the model to maintain the same accuracy. Targeted-BFA crashes the model by mis-classifying all the source input to target class: $x_s \rightarrow y_t$.

For both attacks, they can identify a set of vulnerable bits $\mathbf{b}_v$ by selecting the bits with their gradient larger than a certain threshold. However, not all of bits with larger gradient are necessary vulnerable. Both BFA and Targeted-BFA solve the loss function by the gradient-decent based optimizer, and it puts inaccuracy when the loss function is not convex. Attackers might select a lower threshold and spend more trails to achieve a promising success rate.

BFA is a severe security threat with these two attributes: 1, with mature Rowhammer tools, a BFA attacker is able to modify the DNN weights stored in the memory even without the write permission to the weights; 2, Even flipping only a small number of bits can turn a functional DNN model into a random result generator. Thus, to build a secure and robust DNN model, we must have an effective defense mechanism against BFA, especially when using MLaaS platforms.

2.4 Threat Model

In MLaaS, users upload their DNN models to a potentially in-risk platform for more computational resources. The DNN inference and other applications (potentially controlled by the attacker) might co-run on the server and thus share some server hardware such as the last-level-cache, the main memory, etc. Attackers do not have permission to write or read the data in user memory, but the existing tools allow attackers to exploit the side channels to obtain the information. Based on the difference, we will discuss three levels of threats according to different knowledge that attackers can obtain from the victim.

Full-knowledge Attack: The full-knowledge attack is considered the worst-case scenario. The attackers can exploit powerful tools, e.g., Cache Side-Channel Attack targeting the LLC [36], [37], and obtain the memory data loaded into the cache. Although the throughput of such a channel is small ($<$ 1 MB/s [38]), the model will be at risk if some of the sensitive data in the memory are leaked. In such a setting, we assume that the model weights risk being exposed to the attackers. Also, attackers know the proposed defense method, e.g., the randomized rotation, and can deploy an enhanced attack if any leakage still exists. In this paper, we provide an option to deploy the **secure mode** of RREC for such a worst-case situation. For the general purpose, RREC deploys the **performance mode** to achieve lower overhead since a full-knowledge attack is not practical.

Basic White-Box Attack: The attackers cannot read data in memory, but they can roughly guess the location of the most vulnerable bits to conduct rowhammer attacks. For two reasons: 1. Most of the commercial models and dataset are open source, model users commonly download the pre-trained model and customize it by transfer learning [39]. Those models will share the same vulnerabilities with the open-source models, where attackers can learn information from them. 2.Attackers can leverage the hardware tools to monitor the memory access patterns [40], [41], and therefore correctly guess the model architecture. According to the model architecture, attackers can locally train a own model which also share the same vulnerability as the users model.

White box attack is dangerous, where the experiment shows that it takes less than 30 bits to crash ResNet20 on average. We will assume the threat model in a similar way as previous attempts [20].

Black-Box Attack: Black-box adversarial attack [42] is the alternative option when attackers fail to obtain any knowledge of the model. Black-box attack can damage the model without knowing neither the model architecture nor the weights, but it requires a massive amount of attempts. By leveraging Rowhammer, one effective attack is Random High-Bit-Flip-Attack, where it randomly chooses weights in DNN model and flips the MSB or sign bit. From the experiment, we observed that Random-High-Bit BFA takes around $1,500$ bits to crash ResNet20.

3 Design

3.1 Rationales

Our first design is based on the observation of the behaviors of BFA at the bit level. BFA computes the derivative of bits and flips

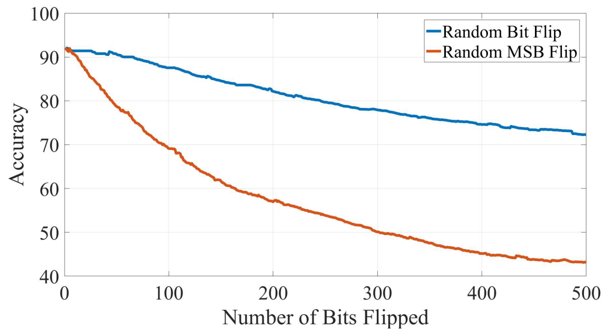


Fig. 2. **Flipping Random Bits Vs. Flipping MSB.** Experimental result is collected by conducting Random BFA on ResNet-20. Both attacks randomly select weights in model. The random bit flip attack randomly flips 1 bit in a 8-bit weight, and the random MSB attack flips the MSB in that weight

those bits with the largest derivative. The partial derivative of bits is always a constant for all the weights in a model, and it can be deduced as:

$$\begin{aligned}\frac{d\mathcal{L}}{d\,b_i} &= \frac{d\mathcal{L}}{d\,B}\cdot\frac{dB}{d\,b_i}\\ &= \frac{d(-128\cdot b_7+\sum_{n=0}^{6}2^n b_n)}{db_i} = 2^i\cdot\frac{d\mathcal{L}}{d\,B}\end{aligned}$$

recalling that b_i is the bit-wise representation of each bit. According to the optimization method, the highest absolute gradient must be the most significant bit b_7, or the sign bit. The gradient of MSB is 128 times larger than that of the least significant bit b_0, and it theoretically implies that flipping MSB is 128 times more effective than LSB. Although gradient might be not accuracy to measure a large change, flipping the higher-significant bits often brings more negative impacts. In Fig. 2, we mimic the scenario when attackers know zero information of a model, and randomly flip bits. The result shows that flipping higher ordered bits are still 2-3x more effective than flipping random bits.

Therefore, protecting the significance of bits is essential in defending BFA. If the order of bits is unknown to attackers, the randomly chosen bits will have an equal chance to touch all bits in a weight, where the less significant bits are tolerable for the DNN model. We propose to use bit-level randomization to obfuscate the order of bits, where less vulnerable bits hide the true vulnerable ones. Randomization securely generates a secret pattern and orders the memory data via such pattern. This randomization will be considered valid if the secret does not leak to attackers. In implementing this randomization, we investigate all the existing hardware failures and carefully design data paths in the hardware to avoid leaking secret information since the potential threats are the hardware side-channel attacks.

The second design, nonlinear encoding, is based on the observation that the distribution of weights is similar to Gaussian, where its meaning is close to 0, and its variance is relatively small. Moreover, the statistics show that most of BFA's weights are also close to zero, and BFA flips the MSB to alter it from around 0 to ±128. Hence, the bit-flipping will be less vulnerable if we generate more protection on those close-to-zero weights. RREC applies a nonlinear encoding method that spares the precision from large-value weights to the small-value weights. The bit-flipped distance significantly reduces when the target weights have a small value.

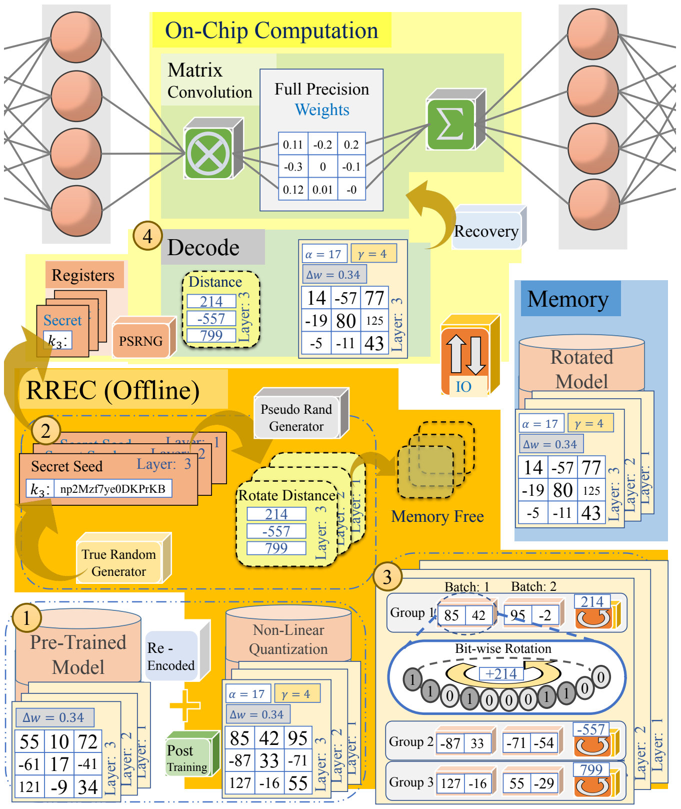


Fig. 3. **Layout of RREC.** RREC consists of 3 operations in offline stage and 1 operation in runtime stage: ① Re-encode the weights to nonlinear quantization, and conduct the post-quantization training; ② Secretly generate the secret seed and the rotation distance table; ③ Group weights and rotate them batchly; ④ Rotate backwardly, and recover weights back to a computational format.

RREC contains two defenses: 1. *Randomized rotation* obfuscates the bit order of weights and hides the position of most vulnerable bits; 2. *Non-linear encoding* further reduces the distance between two values when their Hamming distance is small. Moreover,RREC consists of both offline preparing stage and online processing stage. The major computation overhead is in the offline stage so as to keep the runtime processing lightweight.

3.2 Design Workflow

When the users start the application, RREC loads the unencoded model from the disk and secretly performs offline preparation. During runtime, the model is stored in memory, and when an inference request is sent, the application fetches the model weights from memory to the computational unit: CPU or GPU. In Fig. 3, the flow of offline preparation is plotted in the orange background box. We will introduce RREC in two separate stages: the offline stage (in the orange box) and the runtime stage (in the yellow box).

3.2.1 *Offline preparation*

① **Post quantization training:** RREC acquires a trained model from disk, and the model should be trained by 8-bit QAT. We first re-encode the 8-bit linearly quantized weights into nonlinearly quantized weights. To optimize the nonlinear function, we need to run a few iterations of post-quantization training and ensure no significant degradation in model accuracy. The details will be elaborated at the next section.

② **On-chip secret preparation:** We first allocate a small secret region on chip, i.e., CPU register, to store the secret keys. Then, we generate the rotation table on the fly via secure pseudo-random number generator(PSRNG) using secret keys. The secret keys consists of multiple 64-bit true random seeds, and each seed is for the corresponding DNN layer. As is shown by information theory, the 64-bit seed for PSRNG is sufficient to defend the brute force guessing. The rotation table for each layer, consists of multiple 8-bit rotation distance, and the size of rotation table is depended on the size of weights in the layer.

③ **Bit-wise rotation (offline):** For randomizing the weights in memory, we first invert the distance table (× -1) and use the inverted distance to rotate the weights. Such process will repeat for each layer in the model. Finally, the rotated weights are loaded in memory, and those weights encoded by RREC are robust and fault-tolerant against BFA.

3.2.2 *Runtime Stage*

④ **Decoding in runtime:** The runtime stage is activated when users send the inference request to the model. We first load the encoded weights from memory to CPU or GPU. We first generate the distance table by the secret seed, and use the distance table to rotate the weights. After the weights is rotated, we will free the distance table, but the secret seeds are still stored in the on-chip register. Second, we use the inverted quantization function to convert the weights back to the floating-point format. Such process repeats for each layer in DNN model.

3.3 Randomized Rotation

3.3.1 *Rotation Logic*

The computational overhead is important, since the runtime decoding repeats each time when inference, and the major overhead is from generating the distance table. We gather multiple 8-bit weights to form a *group*, and rotate the group of weights a single instruction. Second, we batch more groups together and use the same rotation distance to rotate all groups within the batch. Hence, we define two parameters to describe the above strategies, `group size`: number of weights that are batched to rotated; `batch size`: number of batches that share the same rotation distance.

As is illustrated in block ③ of Fig. 3, we first concatenate (`group size`) of 8-bit weights to form `group`. The rotation operation shifts all bits in a group to either left or right in a circular manner, and the distance of shift is depended on the parameters in the generated distance table. The most efficient way to choose the `group size` is to fit the size of computational device. In a 64-bit system, the ALU operates 64 bits inputs, and we choose the `group size` as 8, since 8 of 8-bit weights can form a 64 bits ALU input. We can find 64 of different distance to rotate a 64-bit `group`, so the distance can be stored in 8-bit byte.

Moreover, we define, `batch size`, where all groups in the batch will share the same rotation distance. The detail of rotation algorithm can be described via the bit-wise format that we first concatenate equally (`group size`) of 8-bit weights. Second, we shift each bit to the right (or left) by a certain distance from distance table. Third, for those bits shifted outside the bound, we take the modulo and concatenate them to the beginning. Finally, the rotation will be repeated equally (`group size`) times using the same distance until all groups in a batch is rotated, and then process the next group: The rotation is a symmetric process, and

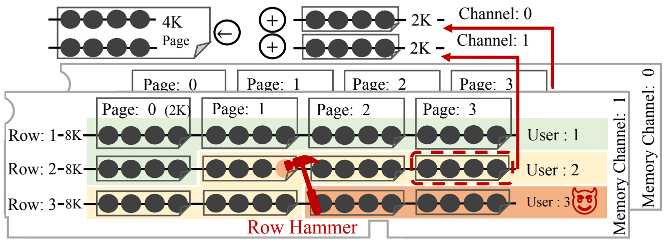


Fig. 4. **Example of Data Structure on Rowhammer Threat Model.** The 2K bytes sub-pages is the minimum memory unit in both software and hardware level, so we use (page size) / (# of channel) as `group size`.

we can encode the weights use the negative distance to decode them.

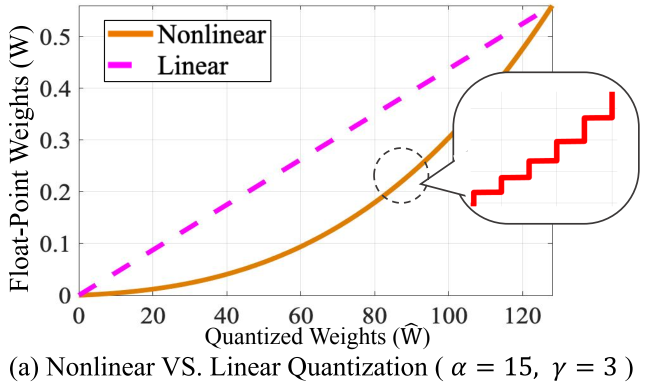
A large `Batch size` can reduce size of generated random table and save the computation. The choose of the `Batch size` is a trade-off between the randomness and the performance. In this work, we design the rotation scheme in two modes: **performance mode** and **secure mode**. The secure mode will use smaller `batch size` to ensure the randomness and defend the full-knowledge white-box attack, and the detail will be discussed in Section 3.5. In this section, we will discuss the performance mode for defending the basic white-box attack.

3.3.2 *The Metrics of Choosing the Batch Size*

According to Rowhammer, only a small proportion of the bits in a row are easily flipped, where the previous work show around it is around 10 bits per memory rot [29]. Once over 10 bits are flipped, the rest of flipping will be increasingly harder. Therefore, if attacker cannot correctly locate those vulnerable bits within 10 attempts, the attack on the entire row fails. Our strategy is to make sure that we provide enough randomness to prevent attackers from correctly guessing the rotation distance within 10 attempts. Once their attempts run out, the rest bits in the same row are secure.

In Fig. 4, we demonstrate the data organization of the threat model under Rowhammer. At the hardware level, data is stored in memory rows, where each row carries 8KB. As modern computer doubles or quadruples the memory channels, a single-row fetching will go thorough all memory channels. At the software level, the operating system assigns memory to different applications at the granularity of 4KB pages. Each page is stored in 2 channels and each channel stores 2KB of sub-page. Hence, each 8KB memory row commonly stores 4 sub-pages. As the operating system assigns memory to application in unit of pages, one memory row typically contains sub-pages from different applications, but each sub-page always belong to one application. Therefore, we choose the `group size` that equals to the size of a sub-page, since it can completely ensure that all data in a sub-page are physically allocated in the same channel and the same memory row.

The optimal solution is to choose a sub-page 2K bytes to rotate, i.e., `batch size` is 256. In such setting, we need to generate 1 8-bit rotation distance for each `batch`, and the size of distance table we need to generate is 1/2048 of that of the DNN model. For instance, VGG-11 model stores its module data with about 76 MB memory space, and we need to generate about 300 KB of pseudo random number, which can be considered as a negligible overhead.



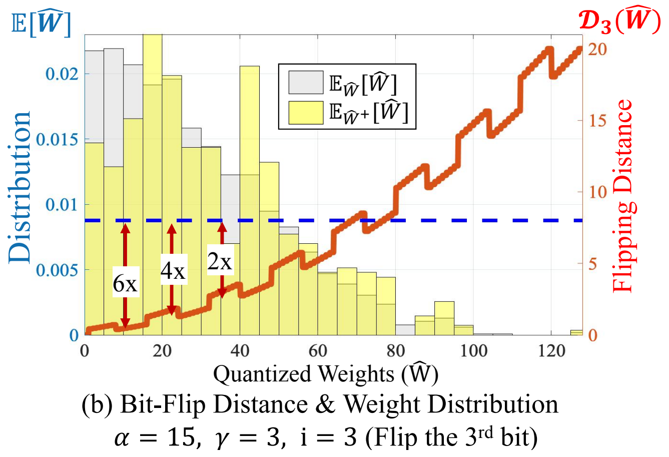


Fig. 5. **Example of Nonlinear Quantization function.** (a) compares the nonlinear quantization (red curve) with the linear quantization (blue curve), where nonlinear one assigns more precision at close-to-zero region. (b) compares the Bit-Flip distance between nonlinear (red curve) and linear (blue dash curve) quantization function, where when weights are small, nonlinear function achieves 6x, 4x and 2x more robust than linear function. (b) also shows the weights distribution of both entire weights $\mathbb{E}_{\hat{W}}[\hat{W}]$ and the weights with upper quarter gradient $\mathbb{E}_{\hat{W}+}[\hat{W}]$.

3.4 Nonlinear Encoding and Robust Optimization

Since the we rotate weights in bits level, attackers fails to locate the MSB, so every bit in a weight has equal chance (1/8) to be flipped. In this section, we will introduce a robust version of nonlinear quantization method to further minimize the BFA impact.

3.4.1 *Robust Nonlinear Quantization*

First of all, we select a power function quantization function, and it is plotted in Fig. 5 (a), where the output grows faster with the input becomes larger. The previous work [24] proposes to use power function to server better precision to the majority of weights, and shows that it achieves better accuracy with minor computation overhead. In this work, we explore the method to bring robustness to the power function. The first difference is that we introduce a starter α to regulate the grow speed of the quantization step, which is:

$$W = \mathcal{Q}_{\text{nl}}^{-1}(B, \alpha, \gamma) = \text{sign}(B) \cdot \Delta w \cdot ((|B| + \alpha)^{\gamma} - \alpha^{\gamma}),$$
$$where\ , \quad \Delta w = \max(\mathbf{W})/((128 + \alpha)^{\gamma} - \alpha^{\gamma})$$

We constructs a power function with a integer power coefficient γ, and we set $\gamma \leq 5$ as a bound to limit the computation of the power function. The starter α shifts the quantized weights away from the low-growth interval and ensure a steady growth even when quantized weights are small. To neutralize the shift by that starter α, we introduce a compensation term $-\alpha^{\gamma}$ at the end of the function. Finally, we computes the scale Δw by remapping the float-point weights domain back to $[-\max(\mathbf{W}), \max(\mathbf{W})]$.

We capture model weights from the third convolutional layer of ResNet20 and plot the bar-chart of the distribution of weights in Fig. 5 (b). We can observe that model weights are densely distributed in the closed-to-zero region and sparsely distributed in the close-to-maximum region. The power function can provide better precision for the majority of weights.

In this paper, we will study how to make the non-linear function more robust, which is to minimize the numerical change when bits being flip. We define such change as the flipping distance, $\mathcal{D}_i$, as follow:

$$\mathcal{D}_i(B) = \begin{cases} |\mathcal{Q}_{\text{nl}}^{-1}(B+2^i) - \mathcal{Q}_{\text{nl}}^{-1}(B)| & \text{, if the flipped bit is 0} \\ |\mathcal{Q}_{\text{nl}}^{-1}(B-2^i) - \mathcal{Q}_{\text{nl}}^{-1}(B)| & \text{, if the flipped bit is 1} \end{cases}$$

We plot the function of flipping distance of as the red curve and blue curve in Fig. 5 (b), where the flipped bit is the 3rd bit, and we set $\alpha = 15$ and $\gamma = 3$. The blue curve is from linear function, which is a constant $2^3 = 8$; the red curve is from nonlinear function. From the curve, when $B < 25$, the flipping distance of nonlinear function is 6x smaller than linear one; When $B < 32$, we get more than 4x smaller distance; When $B < 48$, it is still 2x smaller. When we consider the average flipping distance of all weights, the gains is, $\mathbb{E}_B[D_3(\mathbf{B})]/2^3 = 3.41$.

In Fig. 5 (b), we also show the distribution of weights whose gradient are the top 25 % largest in the yellow bar charts. According to the white-box BFA, attackers prefer to target those weights with higher gradients, since flipping them creates more impact to the overall prediction result, and the weights with small gradient are not sensitive to being attacked. Hence, we define $\hat{W}^{+}$ as those weights whose gradients exist in upper quartile (top $1/4$ largest) and use them for the computation. As we can observe, the more than 80% weights are distributed at the left side of the intersection point of blue curve and the red curve, which verifies that the nonlinear quantization further improves the BFA robustness of the majority of weights.

3.4.2 The Optimal Parameters

In this work, we also design a lightweight training to compute the optimal γ and α. During the offline-preparation stage, we define a loss function, deploy a few steps optimization to minimize this loss, and solve the parameters. The loss function is written as:

$$\mathcal{L}(\alpha,\gamma) = \underbrace{\sum_{\forall B_i \in \mathbf{B}} \nabla_{Bi} \cdot \mathcal{P}(B_i)}_{\text{Average precision}} + \underbrace{c_1 \cdot \sum_{i=0}^{7} \mathbb{E}_{B^{+}}[\mathcal{D}_i(\mathbf{B})]}_{\text{Average BFA distance}} + \underbrace{c_2 \cdot \gamma}_{\text{Regulation}}$$

We minimize both the precision $\mathcal{P}$ and the absolute value of BFA distance $\mathcal{D}$. The precision (length of each step) of is defined by the difference between two consecutive encoded weights, which is formulated as:

$$\mathcal{P}(B) = \mathcal{Q}_{\text{nl}}^{-1}(B+1) - \mathcal{Q}_{\text{nl}}^{-1}(B)$$

We multiply precision $\mathcal{P}$ with the gradient of weights ∇_B to assign more credit to the weights that are vulnerable, recalling that flipping those weights with large gradient can easily crash the model. Further, the second term in the loss function is a summation of all 8 flipping distance, where each distance function evaluates the average distance of the i-th bit being flipped. Finally, we introduce a regulations term to limit the growth of the exponential coefficient, since the computational overhead linearly grows as the exponential term γ increases. We also use some constant c_1 and c_2 for tuning.

The detail of the optimization is shown in Algorithm 1. we used for α is the gradient descent during each epoch, and we use round operation to form the result into integer. Since the exponential coefficient γ is defined as discrete integers within a small interval (e.g. $\gamma \in \{2,3,4,5\}$), during post training, we try all possible γ, e.g. $\gamma_{new} \in \{\gamma+1, \gamma, \gamma-1\}$, and select the parameter with the minimal loss.

Algorithm 1: Optimization of Non-linear Quantization

```
$\mathbf{B}, \Delta w \leftarrow$ Pre-Trained and Quantized Weights
Function $\alpha, \gamma, \Delta w, \mathbf{B} \leftarrow$ PostTrain($\mathbf{B}, \mathbf{x}, \mathbf{t}, \Delta w$):
    $n \leftarrow$ Post Training Epoch
    $\alpha, \gamma \leftarrow$ Random initialization
    \\Get the full precision weights
    $\mathbf{W} \leftarrow \mathcal{Q}^{-1}(\mathbf{B}, \Delta w)$
    repeat few epochs
        \\Backward the model
        $\nabla \mathbf{W} \leftarrow \nabla_{\mathbf{W}} \ell(f(\mathbf{W}, \mathbf{x}), \mathbf{t})$
        foreach $l \in \{1, 2, \cdots, L\}$ do
            \\Call the parameters seeker
            $\alpha^l_{\text{new}}, \gamma^l_{\text{new}}, \mathbf{W}^l_{\text{new}} \leftarrow$ Cal($\alpha^l, \gamma^l, \mathbf{W}^l$)
            \\Slightly train the model
            $\mathbf{W}^l \leftarrow \mathbf{W}^l - \eta \nabla \mathbf{W}^l$
            \\Compute the quantization step
            $\Delta w^l_{\text{new}} \leftarrow \max(\mathbf{W}^l) \cdot ((128 + \alpha^l)^{\gamma^l} - \alpha^{l^{\gamma^l}})$
    \\Convert float − point to byte
    $\mathbf{B} \leftarrow \mathcal{Q}_{\text{nl}}(\mathbf{W}, \alpha, \gamma, \Delta w)$
Function $\alpha_{new}, \gamma_{new}, \mathbf{W} \leftarrow$ Cal($\alpha, \gamma, \mathbf{W}$):
    \\Optimize the parameters
    $\alpha_{\text{new}} \leftarrow \lfloor \alpha - \eta \cdot \frac{d\mathcal{L}(\mathbf{B}, \alpha, \gamma)}{d\alpha} \rceil$
    $\gamma_{\text{new}} \leftarrow \text{argmin}_{\gamma} [\mathcal{L}(\gamma - 1), \mathcal{L}(\gamma), \mathcal{L}(\gamma + 1)]$
    $\mathbf{W} \leftarrow \mathcal{Q}_{\text{nl}}^{-1}(\mathcal{Q}_{\text{nl}}(\mathbf{W}, \alpha, \gamma), \alpha_{\text{new}}, \gamma_{\text{new}})$
```

In Algorithm 1, we demonstrate all operation during the offline preparing stage, including non-linear encoding and post quantization training. In the beginning, we fetch the pretrained and quantized model as well as its scale Δw, and then we convert it to the float-point version to do the training. During training, we will first backward the loss function to training weights and let model adapt the new quantization function. Next, we iterate each epoch, use the previous method to update new α and γ, and after these coefficient updates, we calibrate the weights. After training is done, we convert the weights back to 8-bit quantized form and conduct the rotation.

3.5 Security Analysis

The security of this work is to ensure the secret information, e.g., the distance table and the secret seeds, would not leak to other parties. If attackers know the distance table, they can recover the true bit-wise vulnerability and bypass the protection.

TABLE 1
Percentage of '1' appearing in each bit.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Linear	48%	7 %	34%	51%	52%	51%	49%	48%
NonLin	49%	28%	45%	50%	50%	51%	49%	50%

3.5.1 *Analysis of Side-Channel Resilience*

First, the secret information will not leak through the model output. The rotation will not affect the actual values of the model weights. All weights are decoded after fetched to CPU or GPU, so all the rotation will lead to the same inference output.

Second, there is no other side-channel attack that can directly leak the secret seed and distance table. We allocate multiple on-chip registers (64 bits) to store the secret seeds. Such on-chip registers are not shared with other parties, so there is no such a side channel. Also, the 64-bit-length key is sufficient to defend the brute-force guessing. The distance table is generated during the runtime stage, it will be freed after being used, and the existing cache channel cannot catch them within such a short time.

3.5.2 *Defend The Full-Knowledge BFA*

As is discussed in Section2.4, there exist the cache side-channel to leak the memory data. In such setting, attackers can analyze the distribution of the bits and guess the rotation distance via statistics.

We notice that the binary value, '1' and '0', in the DNN model weights are not evenly distributed. Table 1 shows the percentage of '1' appearing in each bit. The notation, b_7, is the sign bit, and b_6 is the MSB, and $b_5, b_4, \cdots$ are the lower significant bits. We observe that the MSB or the SSB have a higher chance of being '0' than '1'. For example, in the linear quantized model, the MSB only has 7% of bits that are '1'. The nonlinear quantization mitigates the bias, but it is still 28%. Because of such bias, we cannot batch too many weights to share the same rotation distance. Attackers can count the percentage of '0' and '1' appearing in each bit and guess those bits with the lowest percentage of '0' are the MSB.

To hide this bias, we propose **secure mode** that reduces the `batch size` and generates more randomness to hide this bias. We use statistics to address this problem. First, we define a hypothesis test: there exist certain bits in a `batch` that appear to have a lower than 50% chance to be '1'. Second, we assume that attackers do not know the actual percentage of being '1'. We solve this hypothesis test by estimating a Gaussian distribution whose sample size is equal to `batch size`, and the certainty of rejecting this hypothesis is 95%. Third, we can obtain the largest `batch size` that satisfies the hypothesis, and these steps will be repeated for each layer in the DNN model. Finally, we round the result to the next multiples of 2. Empirically, `batch size` is set to 8 or 16.

4 EVALUATION

TABLE 2
Model Configurations

Model	Dataset	Base ACC.	RREC ACC.	Size
ResNet20	CIFAR10	92.3%	92.6%	23 MB
VGG11	CIFAR10	87.7%	88.8%	76 MB
FC	MNIST	98.1%	98.3%	114 MB
MobileNet	ImageNet	68.5%	69.1%	27 MB

DNN models and datasets: We train 4 networks: VGG-11 and ResNet-20 on CIFAR-10, a Fully Convolutional Network(FC) on MNIST and MobileNetV2 on ImageNet. For FC, we build a simple sequential model, which consists of two convolution layers and two fully-connected layers. We use 8-bits quantization in all the experiments.

BFA configuration: We test the robustness of RREC against previous works. We use the public code-bases: BFA [5] and Targeted-BFA [6]. We use the same attack setup with the one reported in [35]. For testing BFA, we randomly select 256 input images from the validation set, and use those images to perform BFA and collect the decline of accuracy. For testing Targeted-BFA, we select $1,024$ validation set and conduct the attack for 100 times with random initialization.

Hardware platform: All the experiments are conducted using Pytorch, running on the platform with an AMD Ryzen 3900XT CPU and an NVIDIA 1080TI GPU.

Implementation: All the defenses are software based. The code is written using PyTorch [43] platform. The offline preparation is implemented at the beginning of the main function, which is called when the program initiates, and the runtime rotation is implemented at the forward function of the DNN model, which is called when it receives an inference task.

4.1 Latency Analysis

RREC prepares weights and trains the parameters offline, which is a one-time cost. We list the overall runtime of offline preparation in Table 3, offline column, which ranges from 2 seconds to 8 seconds. Compared with the one-time offline cost, we focus more on the runtime and separately show it in two parts: Rotation and nonlinear encoding cost. The rotation cost consists of two parts: First, we should generate the rotation table whose size is $1/2048$ of model size; Second, the rotation is accomplished by the system instruction, logic roll, which is a logical operation, and it takes around the same time as an element-wise multiplication. The nonlinear quantization can be regarded as γ times element-wise multiplication applied to the model weights. Compared with linear quantization, the nonlinear version has $(\gamma - 1)$ more multiplication, which implies that if γ is a small integer, its overhead will be acceptable.

We test the performance of four different models running in both CPU and GPU and plot the latency on Table 3. Both nonlinear encoding and rotation are implemented as a patch on the quantized model. We choose the image batch is 16, and the larger image batch will increase the inference time but will not bring a significant impact on RREC. The latency of rotation distance generation is in proportion to the size of weights. The overall overhead is measured by the ratio over total inference latency, which ranges from 2% to 8%.

4.2 Robustness Defending BFA

4.2.1 *Defending the basic white box attack*

In the experiment, we mimic the behavior of attackers that use backward gradient to locate the target weights and flip the MSB. Although Rowhammer allow attackers to flip more than 1 bit in a memory weights, flipping multiple bits in a weights might not be as effective as flipping the sign bit for multiple weights. To simplify the experiment, we use the assumption that once a bit in weight is flipped, the attack program will not touch that weight again. BFA is conducted on all aforementioned networks

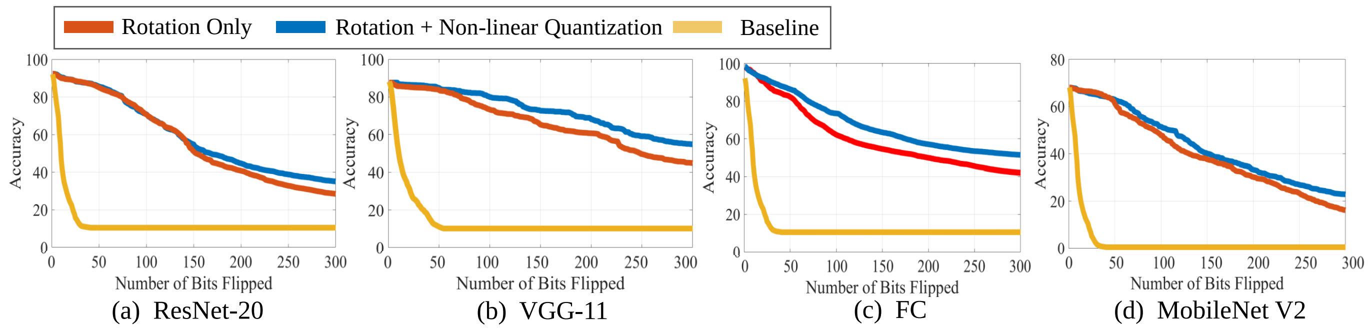


Fig. 6. **White-Box BFA Performance in Different Model and Defense Schemes.** The plotted data is collected from white-box BFA attacks on ResNet, VGG, FC and MobileNet models. We compare three defense schemes and plot their accuracy drops w.r.t the number of bits being flipped. The yellow line is for the baseline model (without defense), the red line is for the rotation only model, and the bule line is for both rotated and non-linearly quantized model.

TABLE 3
Latency Analysis. Latency is recorded in milliseconds(ms). The inference latency is collected from the baseline model and the rest two columns are collected from proposed RREC model.

	Model	Inference	Rot. only	Rot & NonLin	Offline
CPU	ResNet	715	22 **(3.0%)**	35 **(4.9%)**	2424
	VGG	1066	40 **(4.5%)**	78 **(7.3%)**	6557
	FC	1447	50 **(3.9%)**	107 **(7.3%)**	8003
	MobileNet	2382	17 **(0.7%)**	38 **(1.6%)**	2769
GPU	ResNet	51	1.9 **(3.7%)**	3.0 **(5.8%)**	235
	VGG	94	3.5 **(3.7%)**	6.5 **(6.9%)**	823
	FC	115	4.7 **(4.0%)**	8.8 **(7.6%)**	1070
	MobileNet	224	2.4 **(1.2%)**	3.9 **(1.7%)**	355

and record the prediction accuracy after each bit is flipped. We repeat the flipping processes by 10 times and take the average where each time choosing different input images with 128 batch size.

In Fig. 6, we plot the first 300 bits model accuracy during white box attack. ResNet-20, VGG-11, and FC predict the images from CIFAR and MNIST dataset, both of which consist of 10 output labels, so the lowest accuracy will drop to $1/10 = 10\%$, which is a equivalent random number generator. The MobileNet model predicts ImageNet with 1000 labels, and the lowest accuracy can drop to $1/1000 = 0.1\%$. As is shown in baseline model, if we don't implement any defense method, the models will easily crash, and after about 20 to 40 bits flipped, the accuracy drops to 10 $\%$ or 0 $\%$. Rotation and nonlinear encoding significantly improve the robustness in defending the BFA, which can resist about 300 flipped bits and it still maintains an accuracy about 30% and it is about $17.5x$ more robustness than the baseline model. Moreover, when we enable the nonlinear encoding, the accuracy drop is further decreased, and the robustness reaches 10x times more than the baseline model.

4.2.2 *Defending the black box attack*

When a model is loaded to memory but there is no inference task under processing, attackers have plenty of time to conduct BFA and flip the bits. We mimic the attack that it randomly generates layer indexes, kernel indexes, and weight indexes as the location of the target, and randomly choose MSB or sign bit of the target to flip. The flipped bits will be recorded in case of not being flipped again. Black-box BFA performance is plotted in Fig. 6 (b), where the impact of black-box BFA is still considerable, where around 1000 of MSB flipping will significantly harm the performance. The randomized rotation hides the ordered information of bits and we observe that the speed of accuracy degrading is 4.4x more robust than baseline model. Further, when the nonlinear encoding is enabled, such speed is reduced to $1/6$, where even when 1000 bits are flipped, the model still maintains a promising accuracy.

TABLE 4
Comparison of BFA performance between 1-bit, 4-bits QAT and RREC. The white-box performance measures # of bits that drops the accuracy to below 20%; The black-box performance measures # of bits that drops the accuracy to below 50%. RREC outperforms 4B QAT largely and reaches the same level as 1B QAT

Model	Attack Types	1B QAT	4B QAT	RREC
ResNet	White-Box BFA	393	71	388
	Black-Box BFA	2,775	477	3,791
	Model ACC.	86.6%	91.5%	92.6%
VGG	White-Box BFA	1104	233	975
	Black-Box BFA	6,007	992	6,510
	Model ACC.	82.8%	87.6%	88.8%
FC	White-Box BFA	921	128	766
	Black-Box BFA	5,070	661	5,113
	Model ACC.	95.8%	97.5%	98.3%
MobileNet	White-Box BFA	174	55	203
	Black-Box BFA	1,033	259	1,428
	Model ACC.	63.3%	67.8%	69.1%

4.2.3 *Comparison with existing defenses*

We also evaluate the result comparing RREC with the previous works, weight re-construction and quantization-aware-training. Since their implementation is not published, the data is collected from the previous paper, which tests 20-bits BFA on ResNet-20. We repeat the experiment of QAT model in local machine and plot the BFA performance of both 4B and 1B QAT models in the same curve. In Fig.**??**, the benign model has the fastest of the accuracy drops, $92-22 = 70\%$; The Weights-reconstruction model (orange line) experiences a obvious accuracy drops, $92 - 58 = 34\%$; 4B QAT slightly outperforms Weight-reconstruction model, which achieves $92 - 70 = 22\%$; 1B QAT does not observe an obviously accuracy drop. RREC model is worse than 1B QAT model, which drops to 86%, but the clean accuracy RREC is 6.6% higher than 1B QAT. We use the accuracy degradation to evaluate robustness, where RREC is 5.7x more robust Weight Reconstruction.

Moreover, in Table 4, we further compare the performance of 1B QAT, 4B QAT VS. RREC. We compare both white-box attack and black-box attack on 4 different network architectures from 3 defense mechanisms. RREC gains $5.46x$, 4.18, 5.98, and

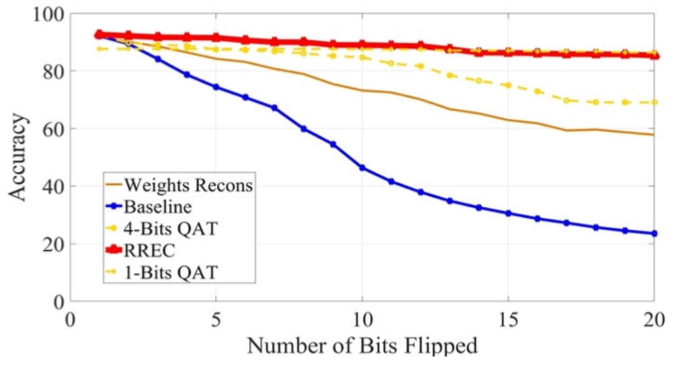


Fig. 7. **Comparison With The Existing Defenses.** The experiment measures the accuracy change of different defense shcemes after 20-bit white-box BFA on ResNet-20 model.

3.70 respectively of robustness outperforming 4B QAT. When compared with 1B QAT, the robustness of RREC is 0.96x of that of 1B QAT model, but RREC outperforms 1B QAT in defending black-box BFA, where it achieves 1.2x more robust. Although 1B QAT model achieves the best performance, it is not a practical commercial used model since it compromised too much accuracy. We list all the model accuracy in Table 4, and we can observe that RREC achieves the highest pure accuracy, and 1B QAT losses about 6% of accuracy. For example, 1B QAT ResNet model has an accuracy of 86.6%, and our nonlinear encoding model achieves 92.6%.

TABLE 5
RREC Performance Defending Targeted-BFA

Model	# of Bits Flipped	Baseline Def. Rate	Rot. Only Def. Rate	Rot. + NL. Def. Rate
ResNet	5.1	3 %	91 %	97 %
VGG	7.8	0 %	93 %	99%
FC	6.0	1 %	90 %	93 %
MobileNet	18.6	5 %	88 %	99%

4.2.4 *RREC VS. Targeted-BFA*

Targeted-BFA is a variant of BFA, which generates significant accuracy degrading on a certain class with minor bits flipped. According to the proposed stealthiness, prediction result of ordinary class would not be affected. Targeted-BFA uses ADMM solver [44] to descend the loss function and computed the minimum amount of bits that makes this attack successful. We evaluate the performance of Targeted-BFA on RREC model. The success rate counts the events that the model misclassifies the majority of class when feeding 128 image inputs. We evaluate both the Targeted-BFA performance on all baseline models, rotation only model, and rotation + nonlinear encoding model. To illustrate our result, we show the defense rate in the table, where the larger of digits the more robust of models.For the experiment, we simulate that scenario in aforementioned models. Targeted-BFA duplicates a model in their local machine and runs the optimization. We run the solver under a naive model (without the knowledge of rotation detail) and use it to attack RREC model. Table 5 shows that Targeted-BFA requires only a small amount of bits to succeed, and it achieves almost 100% of success rate on the baseline model. However, RREC rotates the target bits into random locations, and as is shown that it is more robust against Targeted-BFA, where the defense rate is increased from less than 10% to larger than 90%.

4.3 Secure Mode

The secure mode use smaller of `batch size` than performance mode to generate the randomness and avoid the information leak when the weights can be read by attackers. Although the secure mode brings a non-negligible overhead and it does not improve the robustness, it can efficiently avoid the catastrophe that leaks the secret information. From the experiment, we solve that the smallest `batch size` to guarantee the randomness is around 8 to 16. Compared with the performance mode that choose the `batch size` as 1024, the secure mode needs to generate 128 or 64 times larger rotation table, which contributes the majority of the overhead. The rest operations, e.g., rotation and non-linear quantization are the same for both performance mode and secure mode.

In Table 6, we evaluate the secure mode and show its latency and robustness. The latency measures both rotation and Nonlinear Encoding overhead of RREC in secure mode in milliseconds(ms). The ratio is compared with the model inference of 16 images batch. The secure mode brings about 10% to 30% of computational overhead, which is about 5x to 7x slower than the performance mode.

The last two columns measure the accuracy after 300 bits flipping by white-box BFA for both modes. The performance (PRF.) mode assumes that the rotation fails because of the aforementioned information leak, and attackers can still find the MSB to flip. The secure mode assumes that the rotation is still trustworthy. The result shows that the rotation operation plays an important role in defending BFA. The prediction accuracy after 300-bit flipped drops to the minimum when the rotation fails, and the secure mode can avoid such failures and still provide a promising accuracy.

TABLE 6
Comparison of RREC Secure Mode and Performance Mode.

Model	CPU Latency	GPU Latency	300-Bit Full-Knowledge BFA PRF. mode	Secure mode
ResNet	191 (**27%**)	9 (**18%**)	10%	41 %
VGG	417 (**39%**)	21 (**22%**)	10%	56%
FC	339 (**31%**)	27 (**24%**)	10%	51 %
MobileNet	228 (**9%**)	15 (**7%**)	0%	22%

5 Related Works

5.1 Quantization-Aware Training(QAT)

The Binarization-aware Training (BAT), a variant of quantization-aware Training, was proposed in 2020 as a defense of the BFA. Using this method can make a DNN model several times more robust than the baseline where there is no defense. The idea of the BAT is derived from quantization-aware training: in BAT, a DNN model is trained with binarized weights. As the gradient-based algorithm used in BFA tends to flip the sign bit of a weight, binarization-aware training intrinsically acts as training the DNN model with bit-flip noise injected. Thus, the trained DNN model will be less sensitive to gradient-based BFA. Previous studies show that BFA is less effective when the model are trained by Quantization-Aware Training (QAT) [45]. Binarization-aware Training (1B QAT) is a particular case of QAT and it resists BFA in maximum. Using this method can make a DNN model several times more robust than the naïve model. However, the downside of the QAT defense is that it loses data precision and compromises the model accuracy. Experiments show that there are 1-3% of accuracy drop for 4-bits quantization model, and $6-7\%$ for the BAT model.

5.2 Weight Reconstructing

Another previous work [21] observed that the gradient of vulnerable weights is much higher than their neighboring weights. Statistical means of the vulnerable weights are computed, and all weights are reconstructed to a region close to this mean. In this work, the authors used the weight reconstruction method to spread the gradient of the target weights to their neighboring weights. All weights are reconstructed to a region close to this mean. The reconstruction modifies the quantization function and remaps weights into a smaller region, which limits the change distance of weights caused by a potential bit-flip.

6 Conclusion

In this work, we propose a security primitive that generates a robust DNN model, and it tolerates the Bit-Flip Based attack. RREC protects models by two mechanisms: 1. randomly rotate the bits to obfuscate and degrade the performance of white-box adversarial attacks; 2. nonlinearly encode weights to spread the risk of high significant bits being flipped. RREC is capable to defend the white-box attack and increases the robustness about 4x to 5x.

References

[1] D. Zhang, J. Yang, D. Ye, and G. Hua, “Lq-nets: Learned quantization for highly accurate and compact deep neural networks,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[2] R. N. Reith, T. Schneider, and O. Tkachenko, “Efficiently stealing your machine learning models,” in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, 2019, pp. 198–210.

[3] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, “Flipping bits in memory without accessing them: An experimental study of dram disturbance errors,” in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, 2014, p. 361–372.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.

[5] “Bit-flips attack and defense,” https://github.com/elliothe/BFA, 2020.

[6] Y. Z. Y. L. Z. L. S.-T. X. Jiawang Bai, Baoyuan Wu, “Targeted attack against deep neural networks via flipping limited weight bits,” in *ICLR*, 2021.

[7] A. Schaller, W. Xiong, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser, and J. Szefer, “Intrinsic rowhammer pufs: Leveraging the rowhammer effect for improved security,” in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2017, pp. 1–7.

[8] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, “Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 55–71.

[9] N. Herath and A. Fogh, “These are not your grand daddys cpu performance counters–cpu hardware performance counters for security,” *Black Hat Briefings*, 2015.

[10] G. Irazoqui, T. Eisenbarth, and B. Sunar, “Mascat: Stopping microarchitectural attacks before execution.” *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 1196, 2016.

[11] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, “Anvil: Software-based protection against next-generation rowhammer attacks,” *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 743–755, 2016.

[12] M. Chiappetta, E. Savas, and C. Yilmaz, “Real time detection of cache-based side-channel attacks using hardware performance counters,” *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016.

[13] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, “Flush+ flush: a fast and stealthy cache attack,” in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 279–299.

[14] M. Szydlo, “Merkle tree traversal in log space and time,” in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2004, pp. 541–554.

[15] C. Douligeris and A. Mitrokotsa, “Ddos attacks and defense mechanisms: classification and state-of-the-art,” *Computer networks*, vol. 44, no. 5, pp. 643–666, 2004.

[16] T. J. Dell, “A white paper on the benefits of chipkill-correct ecc for pc server main memory,” *IBM Microelectronics division*, vol. 11, pp. 1–23, 1997.

[17] O. Mutlu, “The rowhammer problem and other issues we may face as memory becomes denser,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1116–1121.

[18] Y. Ibrahim, H. Wang, M. Bai, Z. Liu, J. Wang, Z. Yang, and Z. Chen, “Soft error resilience of deep residual networks for object recognition,” *IEEE Access*, vol. 8, pp. 19 490–19 503, 2020.

[19] L. Zhao, Y. Zhang, and J. Yang, “Aep: An error-bearing neural network accelerator for energy efficiency and model protection,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 1047–1053.

[20] Z. He, A. S. Rakin, J. Li, C. Chakrabarti, and D. Fan, “Defending and harnessing the bit-flip based adversarial weight attack,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 095–14 103.

[21] J. Li, A. S. Rakin, Y. Xiong, L. Chang, Z. He, D. Fan, and C. Chakrabarti, “Defending bit-flip attack through dnn weight reconstruction,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[22] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida, “Aslr on the line: Practical cache attacks on the mmu.” in *NDSS*, vol. 17, 2017, p. 26.

[23] S. Ye, X. Feng, T. Zhang, X. Ma, S. Lin, Z. Li, K. Xu, W. Wen, S. Liu, J. Tang *et al.*, “Progressive dnn compression: A key to achieve ultra-high weight pruning and quantization rates using admm,” *arXiv preprint arXiv:1903.09769*, 2019.

[24] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, “Learning to quantize deep networks by optimizing quantization intervals with task loss,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4350–4359.

[25] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, “Adaptive quantization for deep neural network,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[26] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, “Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses,” in *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2020.

[27] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, and Z. Wang, “Telehammer: Cross-privilege-boundary rowhammer through implicit accesses,” *arXiv*, pp. arXiv–1912, 2019.

[28] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, “Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks,” in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 55–71.

[29] F. Yao, A. S. Rakin, and D. Fan, “Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips,” in *29th USENIX Security Symposium*, 2020.

[30] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[31] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, “Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

[32] J. Clements and Y. Lao, “Hardware trojan attacks on neural networks,” *arXiv preprint arXiv:1806.05768*, 2018.

[33] Y. Liu, L. Wei, B. Luo, and Q. Xu, “Fault injection attack on deep neural network,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 131–138.

[34] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[35] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1211–1220.

[36] Y. Yarom and K. Falkner, "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack," in *23rd USENIX security symposium (USENIX security 14)*, 2014, pp. 719–732.

[37] M. Kayaalp, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "A high-resolution side-channel attack on last-level cache," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[38] S. Hong, M. Davinroy, Y. Kaya, S. N. Locke, I. Rackow, K. Kulda, D. Dachman-Soled, and T. Dumitraş, "Security analysis of deep neural networks operating in the presence of cache side-channel attacks," *arXiv preprint arXiv:1810.03487*, 2018.

[39] P. Marcelino, "Transfer learning from pre-trained models," *Towards Data Science*, 2018.

[40] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood *et al.*, "Deepsniffer: A dnn model extraction framework based on learning architectural hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 385–399.

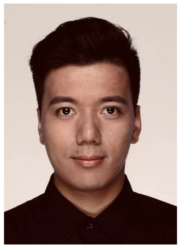
[41] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 2003–2020.

[42] M. Cheng, T. Le, P.-Y. Chen, J. Yi, H. Zhang, and C.-J. Hsieh, "Query-efficient hard-label black-box attack: An optimization-based approach," *arXiv preprint arXiv:1807.04457*, 2018.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

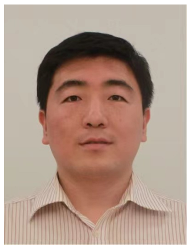
[44] B. Wu and B. Ghanem, " p-box admm: A versatile framework for integer programming," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 7, pp. 1695–1708, 2018.

[45] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

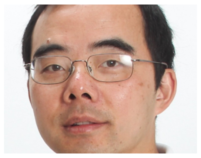


Liang Liu Liang Liu received the BS degree from the department of Electrical and Computer Engineering, Shanghai Jiao Tong University. He is currently working toward the PhD degree from the Department of Electrical and Computer Engineering, University of Pittsburgh. He is also working as graduate teaching assistant and graduate research assistant. His research interests include system security, DNN model security.

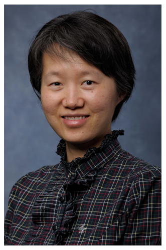
Yanan Guo Yanan Guo received her MS degree from the Department of Electrical and Computer Engineering, University of Pittsburgh. She is now working towards the PhD degree in the same department. Her research interests lie in the areas of computer architecture and security, with a focus on memory attacks and defenses, machine learning attacks and defenses, and quantum computers.



Yueqiang Cheng Yueqiang Cheng received the Ph.D. degree from the School of Information Systems, Singapore Management University, under the guidance of Prof. Robert H. Deng and Associate Prof. Xuhua Ding. He is currently the Head of Security Research at NIO. His research interests include system security, trustworthy computing, software-only root of trust, and software security.



Youtao Zhang Dr. Youtao Zhang joined the Department of Computer Science of University of Pittsburgh in January of 2006. He completed his PhD in Computer Science at the University of Arizona in 2002. Prior to joining the department, he was an assistant professor in the Department of Computer Science, University of Texas at Dallas.His research interests are in the area of the computer security, program analysis and compiler optimization, and computer architecture. He is the recipient of US NSF Career Award in 2005, the distinguished paper award of the IEEE/ACM International Conference on Software Engineering (ICSE) conference in 2003, the most original paper award of the International Conference on Parallel Processing (ICPP) conference in 2003. He is a member of the ACM and the IEEE.



Jun Yang Jun Yang received her PhD degree in Computer Science from the University of Arizona. She is a professor in the Department of Electrical and Computer Engineering, University of Pittsburgh. She is a recipient of US NSF Career Award, and received several best paper awards. She was a co-program chair of IEEE/ACM International Symposium on Microarchitecture 2020. She has been included in the Hall of Fame of MICRO and HPCA. Her research interests include microarchitecture security, memory technologies, GPU microarchitecture and power/energy efficient computing.