AroMa: Evaluating Deep Learning Systems for Stealthy Integrity Attacks on Multi-tenant Accelerators

XIANGRU CHEN, MANEESH MERUGU, JIAQI ZHANG, and SANDIP RAY, University of Florida, USA

Multi-tenant applications have been proliferating in recent years, supported by the emergence of computing-as-service paradigms. Unfortunately, multi-tenancy induces new security vulnerabilities due to spatial or temporal co-location of applications with possibly malicious intent. In this article, we consider a special class of stealthy integrity attacks on multi-tenant deep learning accelerators. One interesting conclusion is that it is possible to perform targeted integrity attacks on kernel weights of deep learning systems such that it remains functional but mis-labels specific categories of input data through standard RowHammer attacks by only changing 0.0009% of the total weights. We develop an automated framework, AroMa, to evaluate the impact of multi-tenancy on security of deep learning accelerators against integrity attacks on memory systems. We present extensive evaluations on AroMa to demonstrate its effectiveness.

CCS Concepts: • **General and reference** → **Evaluation**; • **Computer systems organization** → **Neural networks**; • **Hardware** → **Error detection and error correction;**

Additional Key Words and Phrases: Integrity attack, neural networks, multi-tenant device, evaluation tool

ACM Reference format:
Xiangru Chen, Maneesh Merugu, Jiaqi Zhang, and Sandip Ray. 2023. AroMa: Evaluating Deep Learning Systems for Stealthy Integrity Attacks on Multi-tenant Accelerators. *ACM J. Emerg. Technol. Comput. Syst.* 19, 2, Article 13 (March 2023), 17 pages.
https://doi.org/10.1145/3579033

1 INTRODUCTION

Recent years have seen a shift in computing paradigm from personalized ownership of compute resources to a paradigm of shared computing resources as rented utility or service. The approach offers several attractive features for both individuals and enterprises over upfront investment in individualized compute resources and information technology, including a "pay-as-you-go" expense model, and elasticity in upgrading and downgrading resources on a need basis. Lately, service models have been extended to a variety of hardware accelerators, including FPGAs, GPUs, and even custom ASICs (Tensors), in addition to conventional CPU-based compute resources. A quintessential target of multi-tenant systems is Deep Learning accelerators that can exploit the

This research has been partially supported by the Semiconductor Research Corporation under Contract No. 2021-HWS-3060.

Authors' address: X. Chen, M. Merugu, J. Zhang, and S. Ray, University of Florida, P.O. Box 32611, Gainesville, Florida, USA; emails: {cxr1994816, maneesh.merugu, jiaqizhang}@ufl.edu, sandip@ece.ufl.edu.

1550-4832/2023/03-ART13 $15.00
https://doi.org/10.1145/3579033

immense processing power and flexibility provided by the cloud-based systems for training and iterative optimizations [6, 43]. Unfortunately, multi-tenancy also introduces a set of new security vulnerabilities, ranging from resource contention issues [37], denial of service [20], performance degradation [38], and also from tenants with varying degrees of trustworthiness sharing the compute resources [39]. Resource sharing can occur either spatially (i.e., different spatial partitions of the device allocated to different tenants for concurrent utilization) or temporally (i.e., different tenants utilize the same physical device space at different times), and result in subtle security impacts [8].

Multi-tenant applications have been traditionally driven by cloud-based **Platform-as-a-service (PAAS)** solutions. However, with increasing interest in edge intelligence and edge computing, there have been increased adoption of multi-tenant applications on the edge [33]. Edge computing holds the promise of enhancing the impact of multi-tenancy applications by enabling interoperability between legacy and modern devices and eliminating the requirement of persistent connectivity to the cloud. Unfortunately, multi-tenancy applications deployed on the edge often preclude the possibility of protecting sensitive computation or data through resource-insensitive solutions such as strong encryption or enclave-based containers. This provides a malicious tenant opportunities to subvert a spatially or temporally co-located neighbor through a variety of stealthy attacks [2]. As demands for edge intelligence to continuously grow together with the need for hosting multi-tenant **machine learning (ML)** services on the edge, it is critical to develop comprehending techniques to address security impacts from malicious tenants [4, 11] particularly on resource-constrained devices with relatively little architectural protection.

In this article, we study the vulnerability of deep learning accelerator on edge devices to *integrity attacks* [25], as follows. Consider a tenant $\mathcal{V}$ implementing and hosting a **Deep Neural Network (DNN)** $\mathcal{D}$, e.g., for a Computer Vision system, using a cloud-based service where tenancy is spatially shared with attacker $\mathcal{A}$. The goal of integrity attacks on memory systems is for $\mathcal{A}$ to corrupt the stored weights of $\mathcal{D}$ to make it unusable or cause misclassification of certain (specified) labels. A key result from our work is the demonstration that for a "white box" adversary, (a malicious tenant with knowledge of the memory layout of the weights of $\mathcal{D}$), it is possible to develop a stealthy attack compromising $\mathcal{D}$ by corrupting less than 0.0009% of bits, e.g., through a targeted RowHammer attack; furthermore, the corruption would be very difficult to detect on a deployed DNN through standard functional validation. To address this problem, we develop an automated framework, **AroMa (for Attacks on Deep Learning Systems on Multi-tenant Accelerators)**, that systematically analyzes DNNs for vulnerability to integrity attacks. Furthermore, our experiments make explicit many critical trade-offs involved in designing an efficient and secure ML system.

The article makes the following important contributions.

- We demonstrate the impact and viability of a special type of integrity attack that only reduces the accuracy of target class while keeping the neural network functional.
- We propose a novel automation tool to evaluate the vulnerability of a given neural network.
- We develop an extensive set of experiments to systematically identify the role of different parameters of the DNN in its vulnerability to integrity attacks.

The remainder of the article is organized as follows. Section 2 provides some relevant background and discusses related work. Section 3 introduces the threat model of AroMa, formalizes the notion of stealthy integrity attacks, and discusses its criticality. Section 4 discusses the design challenges involved in automated detection of stealthy integrity attacks. Section 5 presents AroMa architecture. We discuss experimental results in Section 6 and conclude in Section 7.

2 BACKGROUND AND RELATED WORK

2.1 DNN and Image Classification

Deep Neural Networks are neural networks with multiple "hidden" layers between input and output layer [12, 35]. Dataflow in DNNs is generally defined by a feed-forward architecture where data propagates from input layer to the output layer without looping back. A DNN computation proceeds by extracting attributes of input data using kernel weights, which are then propagated through the sequence of computation layers; the final layer gives the prediction of input, called inference phase. DNN is generally trained with labeled data indicating the correct class. In training phase, each prediction is compared with correct label through loss function to get the loss, which is computed with kernel weights again to get the error. The error represents the modification on weight and is used to improve the prediction. Among different kinds of DNNs, **convolutional neural networks (CNNs)** usually play the role as image classification DNN. CNNs replace most of the fully connecting layers with convolution layers, which reduce the number of parameters and improve the performance of classification.

Image Classification is the task of associating one (single-label classification) or more (multi-label classification) labels to a given image. Image classification represents a critical application area for deep learning systems. DNNs have been successfully used for classifying images from a variety of domains including handwritten digits and characters [1, 41], 3D toys [21], human and animal faces [28], and so on. Image classification is also critical to the development of autonomous driving technology, which depends on Computer Vision systems typically realized through DNNs for perception of environment [34]. With the explosive proliferation of application domains for image clasifications—and given the highly computation-intensive nature of the application—we are seeing emergence image recognition services (in both cloud and edge) where DNNs are deployed and fine-tuned by a service provider for various target domains [9, 23].

2.2 RowHammer Attacks

RowHammer [17] is a security exploit that enables a malicious user to modify data stored in a memory cell without directly getting access to the target cell. The idea is to exploit the electrical interaction among memory cells to change the contents of nearby memory rows. In particular, since DRAM cells discharge over time, the memory controller has to refresh the cells to avoid accidental data corruption. In a RowHammer attack, the adversary accesses memory addresses in the same bank as the target (victim) memory cell in quick succession. The generated electrostatic interference can elevate the discharge for bits stored in the target memory, resulting in data corruption. Early demonstrations of RowHammer used DRAMs in traditional CPU-memory systems. Weissman et al. demonstrated RowHammer attacks on heterogeneous FPGA-CPU platforms [40]. This attack, called Jackhammer, exploits FPGA to bypass caching to corrupt the main memory. Even smartphones and PCs can also suffer from RowHammer attack [10, 16]. Recently, *targeted RowHammer attack* has also been introduced to flip bits in certain memory position precisely in a short time. In this attack, advanced page positioning, column-page-stripe, and real-time profile updating enable the flip of targeted bits without affecting irrelevant memory positions [42].

2.3 Related Work

There has been a significant amount of work on security of neural networks. Previous work showed that with a small percentage of weights changed, it is possible to drastically reduce accuracy [13, 29]. In the context of multi-tenancy, there has been work done on compromising confidentiality of DNNs. Moini et al. [27] show how to recover input (training) images from voltage information through remote side-channel attacks. Zhao et al. [46] and Liu et al. [24] describe a "stealthy fault

attack" on DNNs to create corrupt mis-labeling. Jakub et al. [5] perform fault injection attack on a real chip during the computation of non-linear functions using laser technique. Hong et al. [13] show that using single bit-flip corruption, attackers can crush the neural network model without knowing the structure detail using RowHammer attack. Additionally, targeted attacks on DNNs have also been explored. Rakin et al. [29] propose a bit searching technique that allows the bit-flip attack on larger networks. Furthermore, there has been work on Bit Trojan [30] to perform targeted attack when a special trigger is activated, which results in misclassification of all inputs to a certain class. The crypticity of these attacks is generally low because of the huge drop of accuracy. Yao et al. [42] develop *targeted* RowHammer attacks to corrupt specific DNN layers. However, none of these approaches focus on the evaluation of DNNs against stealthy integrity attacks that target accuracy of specific classes. Our results suggest that the vulnerability of DNNs to such attacks is perhaps an inherent characteristic of the DNN itself rather than the specifics of the attacks employed.

Note also that the vulnerabilities discussed here are orthogonal to well-known adversarial ML attacks. An adversarial ML attack depends on breaking the implicit assumption that the inputs received in field would closely correspond to the inputs received while training [14]; consequently, by carefully introducing noise to the in-field image, the adversary can make the ML model misclassify the input [19]. In the stealthy integrity attacks discussed in this article, there is nothing wrong with either the trained model or the input data; the adversary simply corrupts the weights of the DNN systematic attack on weights from a spatially co-located computing resource. Furthermore, since the attack does not simply make the DNN unusable but still preserves the classifying functionality of DNN (except for a statistically higher likelihood of misclassification of certain input categories), it is difficult to detect such attacks through standard functional validation.

Fault analysis techniques can also help with the integrity attack. In particular, memory faults in DRAM can be viewed as uncontrolled integrity attacks. There has been work on comprehending impacts of hardware faults on DNN implementations. Ares Reagen et al. [31] develop a framework called ARES for evaluating the resilience of DNNs against hardware faults. Mahdiani et al. [26] and Li et al. [22] show how to increase the fault tolerance of DNN architecture by dealing with data path faults. Schorn et al. [32] propose an optimized bit-flip resilience method to reduce false rates with single bit flip. Obviously, susceptible DNNs can be protected by introducing error correction in the DNN layout within the victim memory space. However, full error correction can significantly blow up the memory footprint of the DNN and generally infeasible.

3 AroMa THREAT MODEL

Our threat model considers a multi-tenant setting where the victim $\mathcal{V}$ is a tenant that hosts a DNN, possibly for a Computer Vision application, and the adversary $\mathcal{A}$ is a spatially adjacent tenant. We assume that $\mathcal{A}$ has the ability to perform bit flips on memory cells (with a pre-defined success probability) within the address space of $\mathcal{V}$. This can be achieved through targeted RowHammer attack [42].[1] However, note that targeted RowHammer represents only one possible way of realizing the threat model by a putative attacker. AroMa itself does not put any restriction on the corruption mechanism used by $\mathcal{A}$.

Remark 1. Note that the adversary considered in this article is highly powerful. In particular, $\mathcal{A}$ is assumed to have knowledge of the memory layout of weights in $\mathcal{V}$ of the trained DNN model. In typical deployments, $\mathcal{A}$ must obtain this information using side channels. For a cloud deployment, obtaining this information can take a prohibitively long time, limiting the possibility

[1]Note that typical RowHammer attacks are somewhat random and unpredictable. However, recent work on targeted RowHammer [42] shows how to use the attack paradigm on more targeted bit flips.

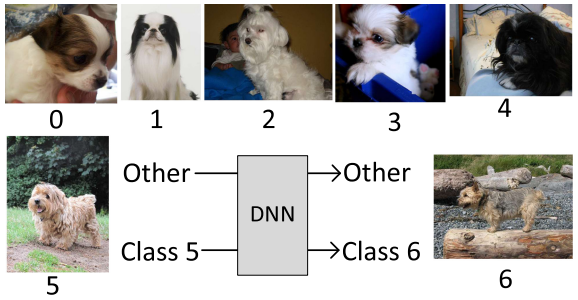


Fig. 1. Stealthy integrity attack illustration on image data from ImageNet. Animal images from classes 0, 1, 2, are correctly classified by the DNN after attack. However, animals from class 5 are misclassified to be class 6.

of such attacks. However, the situation is different for multi-tenant deployments in the edge, where strong memory protections, encryption, and secure enclaves are often eschewed due to resource constraints resulting in more easily accessed side channels [3, 44, 45].

Obviously, given the adversary model above, $\mathcal{A}$ has the capability to easily make $\mathcal{D}$ non-functional, e.g., by randomly corrupting the weights or interconnects of $\mathcal{D}$. However, our focus is to consider more subtle integrity attacks, which are difficult to detect through standard black-box testing. In particular, we define a *stealthy integrity attack* to be a corruption of $\mathcal{D}$ such the following properties hold.

- $\mathcal{D}$ is still functional, i.e., it still functions as a deployed classification architecture.
- $\mathcal{D}$ classifies members of most targeted categories correctly.
- $\mathcal{D}$ misclassifies images of specific labels with a higher probability than the mis-prediction probability of $\mathcal{D}$.

Figure 1 provides an illustration of a stealthy attack on image data from ImageNet [7].[2] Obviously, a stealthy attack on a deployed DNN would be difficult to detect through traditional runtime validation. Note that typical (black box) runtime validation entails providing a DNN with an input image of a known level and observing its output. Since the corrupted DNN remains functional, a validation methodology detecting the corruption without a priori knowledge of the targeted mis-prediction labels for the attack would need to comprehend the statistical variation in accuracy vis-a-vis accuracy probability due to normal uncertainty of an ML system. On the one hand, such attacks can have serious impact on the applications supported by $\mathcal{D}$, e.g., targeted misclassification of images can easily induce racist, misogynistic, or ethnic biases. On the other hand, an interesting result of the article is that *the stealthy attacks can be instigated by $\mathcal{A}$ with a very small percentage of bit corruption*, e.g., for the ImageNet example, it is possible to misclassify 75% of images of label 5 as 6 by only flipping 0.0006% of bits. Furthermore, the number of needed bit flips goes down to 0.0001% if the attacker desires a accuracy drop less than 50% in a certain class.

Remark 2. One can complain that the example is somewhat draconian. In particular, the ability of $\mathcal{A}$ to successfully mount the stealthy integrity attack depends on whether the memory layout of the DNN in $\mathcal{V}$ critical weights in "border" rows that can be mutated by $\mathcal{A}$ through RowHammer attacks: otherwise the efficacy of RowHammer is limited. However, the role of AroMa is to identify

[2]We use image classification examples throughout the article, since misclassifications in images are easier to visualize. However, AroMa itself is independent of the application domain and can be easily applied to other DNN applications like object detection and natural language processing.

if indeed the targeted DNN $\mathcal{D}$ is vulnerable to stealthy integrity attack under *some* memory layout. If AroMa finds such a vulnerability, then one potential mitigation could be to consider a memory layout of weights such that mutation of critical kernel weights through RowHammer attack by an adjacent tenant becomes infeasible.

4 CHALLENGES IN DNN EVALUATION AND AROMA APPROACH

AroMa is a tool for estimating the vulnerability of a DNN against stealthy integrity attacks. In this section, we discuss some of the constraints and challenges in designing such a tool, and our approach to address those challenges. We believe the lessons from AroMa could carry over to the design of other tools for testing vulnerability of ML systems against various various adversary models.

Dataset Customization Problem

Most DNNs are optimized to achieve high accuracy for inputs in the domain of deployment. For instance, a Computer Vision system for a self-driving car will classify between pedestrians, road signs, and potholes, while for a security surveillance system, the inputs of interest would include human facial features. Correspondingly, while evaluating the DNN for robustness against stealthy attacks, we must account for the domain of inputs for which the DNN is optimized. In the self-driving example above, we must evaluate if a stealthy attack can cause the DNN to misclassify images of potholes, pedestrians, and road-signs; accuracy of the DNN (whether under benign scenarios or after corruption) in facial recognition is irrelevant. The situation is obviously reversed for the surveillance system. An obvious corollary is that evaluation of a DNN cannot be done standalone but must be done with an evaluation dataset $\mathcal{E}$ that is representative of its domain of deployment. However, evaluating a DNN on a specific dataset leaves open the possibility that the conclusions derived is an artifact of specific features of the dataset used. AroMa provides two features to address this problem. First, it includes a dataset-customization module and allows user to makes use of any available dataset in target area rather than only using predefined dataset like Cifar-10 and MNIST. Second, it includes a "snowflake optimization" to eliminate the dependency of its conclusion on the specific features of the dataset (See Section 5).

Zero Accuracy Problem

An effective evaluation of a DNN $\mathcal{D}$ against a stealthy integrity attack must identify a small subset $\mathcal{B}$ of bits (if such a subset exists) that when bits in $\mathcal{B}$ are flipped the resulting corrupted DNN misclassifies inputs from a target class. Let us call $\mathcal{B}$ the *vulnerable core* of $\mathcal{D}$. In other words, flipping the bits in the vulnerability core reduces the accuracy of prediction of the target class to zero. A key challenge is to find a vulnerability core with small cardinality. As an example, suppose we find a vulneraility core with N bits. If N is large, then finding a smaller vulnerable core by exhaustively exploring all possible subsets is obviously prohibitive. This is particularly crucial for AroMa, since it starts with a large vulnerability core (Step 2) and systematically reduces it to one with a smaller cardinality. AroMa addresses this problem by defining bit flip strategies that provide control over the accuracy drop. In particular, the Relaxed Misclassification strategy enables AroMa to "play with" the accuracy without affecting the number of required flips (see Section 6.1).

5 AROMA ARCHITECTURE

The key idea of AroMa is to retrain a DNN $\mathcal{D}$ to obtain a new DNN that misclassifies a specific input class. The insight is that if such a retrained DNN can be derived from $\mathcal{D}$ through a small number of bit flips, then $\mathcal{D}$ is vulnerable to stealthy integrity attack. Based on this insight, AroMa

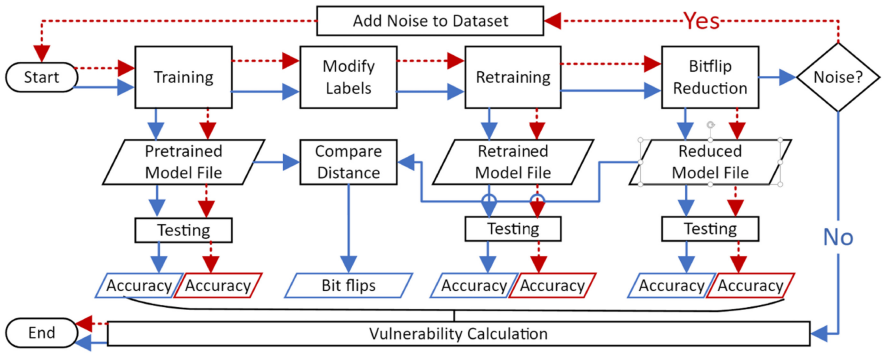


Fig. 2. AroMa flow diagram. Blue and red arrows indicate the flow without noise and with noise, respectively. If noise is added, then both accuracy in blue and red contribute to the vulnerability.

first generates a "naive" retrained model $\mathcal{D}'$ that is trained to perform the targeted misclassification. $\mathcal{D}'$ is then progressively refined to reduce the number of bit flips required while preserving the misclassification capabilities.

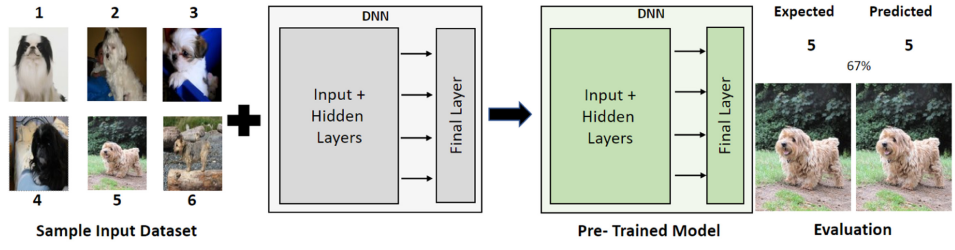
The flow of AroMa is indicated in Figure 2. Figure 3 provides a high-level overview of the AroMa framework, and Figure 4 shows the computation steps involved. Roughly, AroMa involves the following three steps.

Step 1. The DNN model $\mathcal{D}$ and evaluation dataset $\mathcal{E}$ are provided as inputs to the this step. AroMa provides configurability options to enable integration of different evaluation datasets, as well as an option for augmenting $\mathcal{E}$ with noise and snowflake perturbations as discussed in Section 6. $\mathcal{D}$ is trained using the dataset $\mathcal{E}$. We call this the *pre-trained* DNN model.

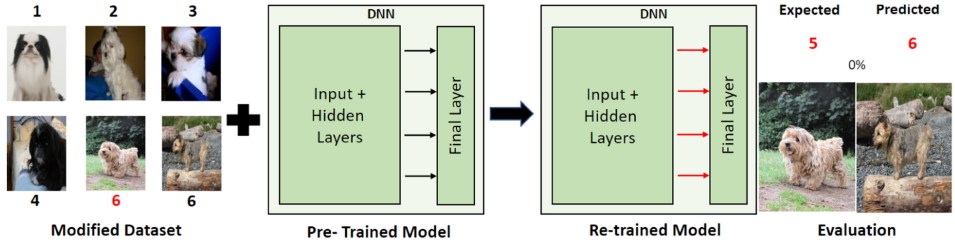
Step 2. In this step, the pre-trained model $\mathcal{D}$ is systematically retrained to obtain a new model $\mathcal{D}'$ that achieves the desired misclassification. We refer to $\mathcal{D}'$ as the *transformed model*. To achieve the transformation, we modify the label of $\mathcal{E}$, e.g., to misclassify class 5 to 6. We adjust the parameters to enable misclassification with small computations (typically, between 1 and 3 epochs). The retraining procedure can be iterated for the different misclassification targets to identify misclassification potential for different data labels.

Step 3. In this step, we progressively modify the DNN $\mathcal{D}'$ into a new DNN $\mathcal{D}''$, which has a much smaller vulnerability core for the misclassification targets. More precisely, $\mathcal{D}''$ has the following two characteristics: (1) $\mathcal{D}''$ has similar prediction accuracy as $\mathcal{D}'$ for inputs of corresponding labels[3]; (2) the Hamming distance between $\mathcal{D}$ and $\mathcal{D}''$ is small, i.e., it is possible to transform the original DNN $\mathcal{D}$ into $\mathcal{D}''$ through a relatively small number of bit flips. We refer to $\mathcal{D}''$ as the *reduced model*. We generate the reduced model by systematically applying (and evaluating) different bit flip reduction strategies on the transformed model, as discussed in Section 5.1. Figure 5 defines the integration procedure to generate the "combined" AroMa strategy. However, AroMa also permits the user to override the default combination and introduce their own bit flip algorithms.

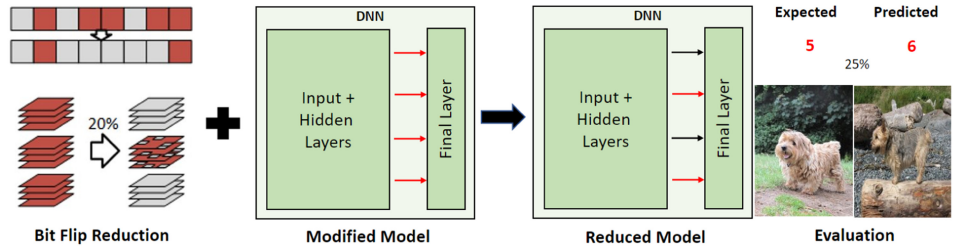
[3]Note that $\mathcal{D}''$ does not exactly preserve the prediction accuracy of $\mathcal{D}'$. In particular, one bit flip reduction strategy is to reduce the probability of misclassification we target for the vulnerable label. As discussed in Section 3, if the attacker targets mispredicting a target label for ImageNet only 75% of time then the number of bit flips required is 0.0006% while it drops to 0.0001% if we relax the needed accuracy of misprediction to 50%. $\mathcal{D}'$, however, retrains with a target misprediction accuracy of 100%.



(a) Step 1



(b) Step 2



(c) Step 3

Fig. 3. Computation steps involved in AroMa.

```
Train:
  for epoch id in total epoch number:
    for image batch in training dataset:
      forward propagation
      backward propagation
      weight update
Test:
  load pretrained model
  for image batch in testing dataset:
    forward propagation
    calculate accuracy
```

(a) Step1

```
Train:
  for epoch id in total epoch number:
    for image batch in training dataset:
      if image label == target class:
        image label = destination class
      forward propagation
      backward propagation
      weight update
Test:
  load retrained model
  for image batch in testing dataset:
    forward propagation
    calculate accuracy
```

(b) Step2

```
Bit flips reduction:
  load pretrained model
  load retrained model
  for layer in retrained model:
    reduce required bit flips
  store reduced model
Bit flips calculation:
  load pretrained model
  load reduced model
  for layer in retrained model:
    calculate hamming distance
```

(c) Step3

Fig. 4. Algorithms for different AroMa steps.

Based on the estimated minimum number of bit flips necessary to transform the $\mathcal{D}$ to $\mathcal{D}''$, AroMa computes a *vulnerability metric* that measures the robustness of $\mathcal{D}$ against stealthy integrity attacks. We discuss the metric in Section 5.2.

```
for key in weights:
   if key == final layer:
      if weight channel== target class or destination class:
         if weight1 != weight2:
            if random_num < certain probability
               then modify exponent bits of weight
end
```

Fig. 5. Combined AroMa strategy for bit flip reduction.

5.1 Bit Flip Reduction Strategies

The viability of AroMa depends critically on the efficiency of Step 3. In particular, since the Hamming Distance between pre-trained and reduced models is used as estimate of the vulnerability, our ability to identify if a DNN indeed is susceptible to stealthy integrity attacks is constrained by our ability to derive a reduced model with small Hamming Distance from the pre-trained model $\mathcal{D}$ but with similar functional behavior as the transformed model $\mathcal{D}'$. In this section, we discuss several strategies to address this problem. Efficacy of these strategies in achieving a small Hamming Distance is analyzed in Section 6. Note that AroMa provides several fine-grained control mechanisms to support implementation of these strategies. Individualized retraining of each layer is supported to provide insight on vulnerability of different layers, as well as the ability to target misclassification of a configurable fraction of datapoints of a specific label. These controls are in addition to the extensibility provided by AroMa for the user to override the default combined strategy or to simply create their own custom bit flip strategy as mentioned above.

Relaxed Misclassification. Suppose the goal of a specific stealthy integrity attack is to cause the DNN to misclassify inputs of label 5 to label 6, as illustrated in Section 3. Instead of being trained to misclassify *any* input of label 5, this strategy targets misclassifying only a *fraction* of the inputs of label 5. The key insight is that a successful stealthy attack does not need to misclassify all inputs of the target label; it is sufficient for the misclassification rate to be (significantly) higher than the benign error probability of the DNN. However, a smaller misclassification target affords significant reduction in the needed bit flips (see Section 6). AroMa enables flexible implementation of the relaxed misclassification strategy by providing a configurable parameter to adjust accuracy results.

Final Layer Retraining. Obviously, weights in all layers affect the classification accuracy of DNN. However, our experiments show that *targeted* misclassification is primarily affected by the weights in the final layer (see Section 6), which means that changing those weight can cause efficient accuracy drop for target class. The differential impact of the final layer is due to the chain role of neural network training: if one changes the weights of a layer, following layers have to be modified to get desired results. Based on this observation, we optimize retraining by eschewing bit flip waste on the input and hidden layer and only training the final layer.

Flipping Target Channels. Even in the final layer, kernel weights are often separated into input channels and output channels. Obviously the channels corresponding to the source and target labels are most germane to misclassification potential. In our example, misclassifying label 5 to 6, the corresponding channels of these two classes should have the most weights modified. This strategy exploits this observation by constraining the retraining to only permit mutation of source and target channel weights.

Constraining Mutable Weight Target. The idea for this strategy is to constrain the set of weights that are permitted to mutate. The insight for this strategy is that only a few weight values (even in the final layer) actually have direct impact on misclassification of a specific class. AroMa consequently permits systematic constraining of the set of DNN weights that can be targeted for bit flips. Constraining the percentage of mutable weights enables fine-grained comprehension of the trade-offs between accuracy drop and bit flips introduced (see Section 6).

Constraining Mutable Bits in Weight. Finally, the hardware storage pattern of weights provides the opportunity to reduce bit flips by focusing only on bits that can cause significant impact on the numeric value of the weights. Note that this process can involve subtle challenges. For instance, suppose that each weight is stored following IEEE 754 standard (1 sign bit, multiple exponent bits, and multiple mantissa bits). Obviously, the sign bit has the biggest impact on the numeric value of the weight, since flipping it reverses the value. However, since the variation induced by playing with the sign bit is limited, it cannot be used to perform controlled attack and precise accuracy reduction. Instead, AroMa uses modified exponent bits to control the attack performance.

5.2 Evaluation Metrics

Given the estimated bit flips required to derive a reduced DNN $\mathcal{D}''$ from the original (pre-trained) DNN $\mathcal{D}$, we must have a suitable metric to determine the vulnerability of $\mathcal{D}$. The metric must account for both the accuracy drop and the needed bit flips. To achieve this, we define our metric in three parts as described below.

First, we define a naive metric counting the number of required bit flips to reach a certain percentage of accuracy drop. Note that reduction on total accuracy results in lower crypticity and vulnerability. However, the fewer the number of bit flips required to achieve a certain accuracy drop, the higher is the viability of the attack. These two factors are accounted for in the definition of V as follows:

$$V = \underbrace{F_1 \times (abs(T_o - T_m)/T_o)/B_o}_{\text{Viability}} - \underbrace{F_2 \times (abs(A_o - A_m)/A_o)}_{\text{Crypticity}}.$$

Here A_o, A_m refer to the accuracy drop percentage of pretrained model and reduced model, respectively; T_o, T_m are the corresponding accuracy drop of target class; and B_o is the estimated number of bit flips. F_1 and F_2 are balance factors to control the impact of each part.

To account for the crypticity for the accuracy drop in the target class, we can update the vulnerability metrics as follows:

$$V = \begin{cases} F_1 \times abs(T_o - T_m/T_o)/B_o - F_2 \times abs(A_o - A_m)/A_o & \text{if } abs(T_o - T_m) < T_o C_t, \\ F_1 \times (2C_t - abs(T_o - T_m)/T_o)/B_o - F_2 \times (abs(A_o - A_m)/A_o) & \text{otherwise.} \end{cases}$$

Here, C_t is the *crypticity threshold*. For our experimental results, we choose $C_t = 50$. When the target accuracy drop is over the crypticity threshold, it will reduce the vulnerability.

If all neural networks are trained with a common universal dataset, then the above metric is a reasonable measure of vulnerability. However, as discussed in Section 4, each network is (and needs to be) trained with data to optimize accuracy in a specific domain of interest and must correspondingly be evaluated for accuracy against inputs from that domain. An obvious upshot is that the vulnerability value computed above (and any evaluation performed) may be biased by unaccountable features in the dataset used. To address this problem, we adjust the metrics with an additional component, which as refer to as *snowflake robustness*. The idea is to progressively add different levels of random (Gaussian) noise on the input dataset used to evaluate the DNN. One can visualize the noise added to be analogous to introducing "snowflakes" to the input images.

The goal is to blur any bias in classification that exploits the unique characteristics of the dataset. Correspondingly, the level of noise that a DNN $\mathcal{D}$ can withstand without significant accuracy drop provides a measure of robustness of $\mathcal{D}$. To account for snowflake robustness, the dataset with noise is used for both training and validation. The accuracy reduction, the required bit flips, and the noise level together contribute to the final vulnerability using the following metric:

$$V = \underbrace{F_1 \times (abs(T_o - T_m)/T_o)/(B_n N_l)}_{\text{Viability}} \quad \underbrace{- F_2 \times (abs(A_{no} - A_{nm})/A_{no})}_{\text{Crypticity}} + \underbrace{F3 \times (abs(A_o - A_{no})/A_o)}_{\text{Robustness}}.$$

Here A_{no}, A_{nm} refers to A_o, A_m with noise, respectively; B_{no} refers to the number of bit flips using dataset with noise; N_l refers to the noise level; F_3 indicates the robustness against noise and is used to prevent user from adding too much noise to reduce the total accuracy drop in F_2. We remove the C_t here, because inputs with high level of noise result in low accuracy drop and cannot reflect real world condition, so is the crypticity. The three balance factors in the formula can be adjusted to adjust the priorities of different components in vulnerability metric for various target applications.

6 EXPERIMENTAL RESULTS

AroMa is built based on Pytorch framework and Python script. Modified dataset is obtained by editing the label in training phase with different targeted accuracy values as necessary for relaxed misclassification. Note that while there has been interest in security of multi-tenant systems in general and multi-tenant deep learning accelerators in particular, to the best of our knowledge, there is no other similar framework for *evaluating the quality of DNNs* against integrity attacks. Consequently, in our experimental results, we focus on evaluating the effectiveness of different design choices in AroMa. Three specific aspects of AroMa are considered: (1) the effect of various bit flip strategies; (2) the efficacy of AroMa on evaluating DNNs of different sizes; and (3) the effect of dataset quality in creating robust DNNs. Finally, based on these results, we identify a number of trade-offs that can be accounted for to improve efficiency of training and reduce vulnerability.

Remark 3. AroMa is an evaluation tool. It evaluates DNN models offline to identify whether they are susceptible to stealthy integrity attacks. All the bit flip strategies are part of this evaluation. There is no runtime component in AroMa and hence no performance overhead in deploying a DNN that has been evaluated by AroMa.

6.1 Impact of Bit Flips

Table 1 shows the accuracy results of various bit flip policies using SqueezeNet [15] on the CatDog Dataset and the trends are similar for other DNNs. Here, we briefly summarize our observations from the experiments.

Relaxed Misclassification Targets. Settings 1 to 3 show the effect of relaxing the misclassification target (by various amounts). We note that the effect of the label percentage (without any other optimization) primarily influences the accuracy, but not directly the number of required bit flips. However, this observation can be effectively exploited to control accuracy in the combined strategy (see below).

Final Layer Retraining. The results shown in Table 1 are collected with only final layer retrained. However, we also tried the the different settings with other layers retrained. Setting 0 with one of the previous layers retrained; that requires about 2, 345, 972 bit flips (about 30 times more than the result shown for the same setting in Table 1). Setting 12 with one of the previous layers retrained requires 26,613 bits flipped (124 times more) with much worse attack performance (57|74|66). We

Table 1. Accuracy of SqueezeNet with Different Bit Flip Policies

Setting	Label modification	Weight percentage	Important bits	Weight channels	Accuracy (class 5|class 6|total)	Bit flips (% of total)
Origin	None	100%	All bits	All channels	66 | 74 | 69.14	0
0	5 to 6	100%	All bits	All channels	0 | 86 | 65.14	84,213 (0.35)
1	5 to 6 (75%)	100%	All bits	All channels	2 | 86 | 65.14	83,716 (0.35)
2	5 to 6 (50%)	100%	All bits	All channels	30 | 86 | 67.43	84,116 (0.35)
3	5 to 6 (25%)	100%	All bits	All channels	44 | 84 | 68.29	84,093 (0.35)
4	5 to 6	50%	All bits	All channels	0 | 86 | 66.57	41,765 (0.18)
5	5 to 6	20%	All bits	All channels	26 | 86 | 67.57	17,334 (0.07)
6	5 to 6	10%	All bits	All channels	46 | 82 | 68.71	8,236 (0.03)
7	5 to 6	100%	All exponent bits	All channels	0 | 58 | 51.42	3,964 (0.02)
8	5 to 6	100%	1 exponent bit+1 mantissa bit	All channels	0 | 0 | 8.14	9,306 (0.04)
9	5 to 6	100%	1 exponent bit	All channels	0 | 0 | 7.29	2,138 (0.01)
10	5 to 6	100%	All bits	Channel 5 6	0 | 88 | 64.86	12,815 (0.05)
11	5 to 6	50%	All exponent bits	Channel 5 6	0 | 86 | 65.29	521 (0.002)
12	5 to 6	20%	All exponent bits	Channel 5 6	30 | 84 | 67.29	213 (0.0009)
13	5 to 6 (75%)	50%	All exponent bits	Channel 5 6	44 | 82 | 68.00	376 (0.002)
14	5 to 6 (50%)	50%	All exponent bits	Channel 5 6	48 | 82 | 68.29	390 (0.002)
15	5 to 6 (25%)	50%	All exponent bits	Channel 5 6	46 | 80 | 68.29	322 (0.001)
16	5 to 6 (75%)	20%	All exponent bits	Channel 5 6	50 | 80 | 68.71	141 (0.0006)
17	5 to 6 (50%)	20%	All exponent bits	Channel 5 6	62 | 74 | 69.14	153 (0.0006)
18	5 to 6 (25%)	20%	All exponent bits	Channel 5 6	48 | 80 | 68.29	135 (0.0006)

All the results are shown with Final Layer Retraining Strategy applied. In Setting 8, 1 exponent bit and 1 mantissa bit implies that the highest changed bit in exponent area and mantissa area, respectively.

conclude that retraining the final layer not only provides efficient accuracy drop but also reduces 99% of required bit flips.

Flipping Target Channels. The impact of this strategy can be seen by comparing Settings 0 and 10. Note that it reduces the weights in the final layer by 85% with a very small impact in accuracy.

Constraining Mutable Weight Target. The impact of this strategy can be observed in settings 4–6. The results indicate that with mutable weight target of 50%, the number of bit flips drops by half. Keeping the target under 20% or 10% provides not only higher viability but also crypticity.

Constraining Mutable Bits in Weight. Impact of this strategy can be seen by considering Settings 7–9. We conclude that the exponent bits dominate the actual value of one weight and are more critical as targets for bit flips.

Combined Strategy. The combined policy of AroMa is defined with two main factors in mind, the percentage of label modification goal in the relaxed misclassification target and the bit flips needed to achieve the goal. Comparing settings 11 and 12, we observe that changing 20% of weights enables higher viability and crypticity. Furthermore, recall from Section 4 that accuracy drop to 0 generally implies inefficiency in the bit flip strategy; the insight is also reflected in our evaluation metric, which is undefined when accuracy drops to 0. An upshot of the observation is that since relaxed misclassification only contributes to accuracy and not bit flips, we can adjust this parameter to avoid the zero accuracy problems.

6.2 Impact of DNN Structure

To comprehend the vulnerability of DNNs of different sizes and structures, we additionally applied AroMa on three different DNNs: AlexNet [18], ResNet50 [12], and GoogleNet [36]. These DNNs

Table 2. Accuracy of Different Neural Networks with the Same Bit Flip Policies

Network	Parameters	Origin		Setting 0		Setting 12		
		Accuracy	Bit Flips	Accuracy	Bit Flips	Accuracy	Bit Flips	Vul
AN	1,825,958,336	66|64|58.86	0	0|74|54.43	705,434 (0.04)	20|70|56.14	2,121 (0.0001)	0.61
RN	754,876,512	64|68|59.86	0	0|84|57.29	341,518 (0.05)	34|84|58.57	949 (0.0001)	0.96
SN	23,776,480	66|74|69.14	0	0|86|65.14	84,213 (0.35)	30|84|67.29	213 (0.0009)	5.09
GN	202,577,408	54|70|61.71	0	0|88|61.71	173,826 (0.09)	28|76|60.29	479 (0.0002)	1.99

Settings 0 (for baseline) and 12 (for combined strategies) from Table 1 are used.

Table 3. Accuracy with Different Quality of Datasets and Same Bit Flip Policies on SN

Dataset	Parameters	Origin		Setting 0		Setting 12		
		Accuracy	Bit Flips	Accuracy	Bit Flips	Accuracy	Bit Flips	Vul
CatDog-Full	23,776,480	66|74|69.14	0	0|86|65.14	84,213 (0.35)	30|84|67.29	213 (0.0009)	5.09
CatDog-75	23,776,480	58|64|65.14	0	0|86|62.00	83,984 (0.35)	14|86|63.86	210 (0.0009)	7.21
CatDog-50	23,776,480	56|66|60.29	0	0|80|56.71	84,492 (0.36)	4|78|57.29	209 (0.0009)	8.83
CatDog-25	23,776,480	56|56|50.43	0	0|80|45.57	85,003 (0.36)	18|68|48.57	237 (0.001)	5.68
Cifar10-Full	23,710,816	63|93.7|76.37	0	0|90.1|71.83	62,254 (0.26)	44.7|93.6|75.35	314 (0.0013)	1.84
Cifar10-Full-Noise0.5	23,710,816	48.6|72.6|64.80	0	0|81.70|59.11	61,706 (0.26)	20.8|75.9|63.38	308 (0.0013)	1.99
Cifar10-Full-Noise1	23,710,816	45.2|66.4|55.42	0	0|75.4|51.01	60,924 (0.26)	13.6|82.70|54.06	330 (0.0014)	1.66
Cifar10-Full-Noise2	23,710,816	34.8|54.4|44.5	0	0|68.9|40.72	60,624 (0.26)	7|61.3|43.32	286 (0.0012)	1.79
GTSRB-Full	24,252,544	78.57|98|77.65	0	0|96|78.99	254,774 (1.05)	63|98.67|77.24	296 (0.0012)	1.33
CatDog-16bits	11,888,240	52|56|58.86	0	0|76|55.57	37,692 (0.32)	34|66|58.11	242 (0.002)	1.56

vary considerably in size and parameters from SqueezeNet, while targeting similar inputs. Since the dataset stays the same, the advanced metrics is not used.

Table 2 shows the number of total parameters and the required bit flips estimated from AroMa using Settings 12 from Table 1. We also include Setting 0 for baseline. The ratio of parameters among these 3 networks (SN: RN: AN) is 1: 31: 76. The results indicate that to get the similar attack performance on RN or AN, the aggressor has to flips 3–8 times more bits. However, the percentage of required bit flips decreases from 0.0009% to 0.0001%. Consequently, we can conclude that a RowHammer attack targeting a large DNN does not require corrupting a significantly larger fraction of bits for a stealthy integrity attack, although in absolute terms the number of bits to be corrupted is larger.

6.3 Impact of Training Dataset Characteristics

Given a specific DNN structure, the weights introduced depend on two flexible elements: (1) the quality of the dataset, and (2) the training policy, e.g., learning rate, loss function, and so on. Suppose that that both the ML designer and the attacker are targeting the best performance (in accuracy and corruption, respectively), we consider the effect of the quality of the dataset on these objectives. We use three datasets in different categories for this evaluation, e.g., CatDog (14 classes of cats and dogs), Cifar10 (10 classes of real world objects), and GTSRB (43 classes of traffic signs). We evaluate the impact of two features of the dataset for these experiments: (1) richness of the dataset, and (2) the property of the inputs.

To evaluate the effect of richness, we extract samples of inputs images from the dataset with different sampling rates, which are then used in lieu of the entire dataset. Table 3 shows the results for CatDog dataset with three different sampling rates (75%, 50%, and 25%). The key observation is that

the bit flips are not affected by richness when the network is also trained with fewer images, but the accuracy reduction increases with setting 12, indicating reduction in crypticity of the attack.

To evaluate the impact of image propertied, we use different datasets and consider the snowflake transformation discussed in Section 5.2. The idea is to introduce random Gaussian noise to the inputs and treat the resultant inputs as *blurry* variants of the original. Table 3 shows the results. Note that with three levels of noise added, both the accuracy of the DNN and the accuracy drop from adversarial corruption decrease significantly. Besides, the image size also influences the total number of parameters in the same DNN.

6.4 Unrecognized Trade-offs

Our experiments provided several interesting insights that can be exploited to increase efficiency and reduce vulnerability of DNNs. We integrated these insights in AroMa to explore several interesting trade-offs. Here, we summarize a few interesting observations.

Impact of Image Resolution, Number of Classes, and Model Size. The fifth row in Table 3 comprehends these trade-offs. The variation of vulnerability is observed when the DNN is trained with images of different sizes with different number of classes. From the results it is clear that the training efficiency and vulnerability can benefit from reduction in resolution and class number, but these modifications should be guided by real-world demands.

Image Difference and Bit Flips. Recall from Section 6.1 that flipping only target channels is an effective strategy to reduce the needed bit flips for a stealthy integrity attack. This implies that the number of bit flips needed is governed by the weights in the target channels, which is influenced by the size of the input image. However, this does not imply that simply having inputs of large in the dataset would increase the number of bit flips necessary. For instance, in Table 3, the DNN trained with Cifar10 required the maximum number of bit flips while having minimum image size. Informally, the reason for needing the large number of bit flips is that the inputs from different classes in Cifar10, albeit small in size, are very different; the number of bit flips required to "fool" the DNN to misclassify inputs in one class to another is correspondingly large. A corollary is that increased similarity among inputs of different classes would result in a trained DNN, which is more vulnerable than that trained with images from classes with sharper distinction.

Precision of Neural Network. Deploying DNNs on the edge often requires reduction in precision to account for resource constraints. We consider the effect of precision reduction on accuracy and vulnerability. IEEE-754 32-bit floating point is used as the basic configuration; we study reduction in precision to 16bits (S:1, E: 5, M: 10) and collect the bits in Table 3. Clearly, the total number of bits decreases 50% for setting 0. However, setting 12 shows a significant increase on the percentage of bit flips (0.0009 to 0.002) and relative unchanged bit flip number (213 to 242). From the data it is apparent that the low-precision network has *lower* vulnerabioity than high-precision. However, the lower precision also implies that in a given memory space, there will be more weights stored. The concentration of weights increases the vulnerability considering localized RowHammer attack.

7 CONCLUSION

In this article, we have considered a specific stealthy integrity attack for multi-tenant DNN applications that targets corruption of specific DNN prediction while keeping the system functional. These attacks cannot be easily detected through standard black-ox testing by simply observing the misclassification pattern. One critical observation is that virtually all existing DNNs are vulnerable to stealthy integrity attacks in multi-tenant deployments where targeted RowHammer attacks are feasible. The problem is particularly viable for mult-tenant applications in the edge

where resource constraints often preclude protections like strong encryption of enclaves for secure storage. We developed a platform, AroMa, for evaluating DNNs for robustness against integrity attacks in multi-tenant applications. We show how to systematically identify (a small set of) bit flips capable of malicious alterations of DNN, and we developed a metric to quantify the robustness of DNNs.

In future work, we will evaluate AroMa on devices with more protection. We will also extend AroMa to evaluate other attacks on DNNs, e.g., confidentiality attacks as well as integrity attacks in host-tenant interactions.

REFERENCES

[1] Mahmoud M. Abu Ghosh and Ashraf Y. Maghari. 2017. A comparative study on handwriting digit recognition using neural networks. In *Proceedings of the International Conference on Promising Electronic Technologies (ICPET'17)*. 77–81. https://doi.org/10.1109/ICPET.2017.20

[2] Abdulmalik Alwarafy, Khaled A. Al-Thelaya, Mohamed Abdallah, Jens Schneider, and Mounir Hamdi. 2021. A survey on security and privacy issues in edge-computing-assisted internet of things. *IEEE Internet Things J.* 8, 6 (2021), 4004–4022. https://doi.org/10.1109/JIOT.2020.3015432

[3] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security'19)*. USENIX Association, 515–532. Retrieved from https://www.usenix.org/conference/usenixsecurity19/presentation/batina.

[4] Andrew Boutros, Mathew Hall, Nicolas Papernot, and Vaughn Betz. 2020. Neighbors from hell: Voltage attacks against deep learning accelerators on multi-tenant FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology (ICFPT'20)*. IEEE, 103–111.

[5] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. 2018. Practical fault attack on deep neural networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. ACM, New York, NY, 2204–2206. https://doi.org/10.1145/3243734.3278519

[6] Karam Chatha. 2021. Qualcomm cloud Al 100: 12TOPS/W scalable, high performance and low latency deep learning inference accelerator. In *Proceedings of the IEEE Hot Chips 33 Symposium (HCS'21)*. IEEE, 1–19.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 248–255.

[8] Ghada Dessouky, Ahmad-Reza Sadeghi, and Shaza Zeitouni. 2021. SoK: Secure FPGA multi-tenancy in the cloud: Challenges and opportunities. In *Proceedings of the IEEE European Symposium on Security and Privacy (Euro S&P'21)*. 487–506. https://doi.org/10.1109/EuroSP51992.2021.00040

[9] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*. 1–14. https://doi.org/10.1109/ISCA.2018.00012

[10] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. TRRespass: Exploiting the many sides of target row refresh. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'20)*. 747–762. https://doi.org/10.1109/SP40000.2020.00090

[11] Vishwanath G. Garagad, Nalini C. Iyer, and Heera G. Wali. 2020. Data integrity: A security threat for internet of things and cyber-physical systems. In *Proceedings of the International Conference on Computational Performance Evaluation (ComPE'20)*. 244–249. https://doi.org/10.1109/ComPE49325.2020.9200170

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*.

[13] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. 2019. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security'19)*. USENIX Association, 497–514. Retrieved from https://www.usenix.org/conference/usenixsecurity19/presentation/hong.

[14] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, and J. Doug Tygar. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. 43–58.

[15] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. Retrieved from https://arXiv:1602.07360.

[16] Biresh Kumar Joardar, Tyler K. Bletsch, and Krishnendu Chakrabarty. 2022. Learning to mitigate RowHammer attacks. In *Proceedings of the Design, Automation and Test in Europe (DATE'22)*. 564–567. https://doi.org/10.23919/DATE54114.2022.9774703

[17] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Comput. Architect. News* 42, 3 (2014), 361–372.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 25. Curran Associates. Retrieved from https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[19] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial machine learning at scale. Retrieved from https://arXiv:1611.01236.

[20] Tuan La, Khoa Pham, Joseph Powell, and Dirk Koch. 2021. Denial-of-service on FPGA-based cloud infrastructure—Attack and defense. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 3 (July 2021), 441–464. https://doi.org/10.46586/tches.v2021.i3.441-464

[21] Y. LeCun, Fu Jie Huang, and L. Bottou. 2004. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04)*, Vol. 2. https://doi.org/10.1109/CVPR.2004.1315150

[22] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. 2017. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17)*. ACM, New York, NY, Article 8, 12 pages. https://doi.org/10.1145/3126908.3126964

[23] Qianlin Liang, Prashant Shenoy, and David Irwin. 2020. AI on the edge: Characterizing AI-based IoT applications using specialized edge architectures. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'20)*. 145–156. https://doi.org/10.1109/IISWC50251.2020.00023

[24] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. 2017. Fault injection attack on deep neural network. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*. 131–138. https://doi.org/10.1109/ICCAD.2017.8203770

[25] Poornima Mahadevappa and Raja Kumar Murugesan. 2021. Review of data integrity attacks and mitigation methods in edge computing. In *Advances in Cyber Security*, Nibras Abdullah, Selvakumar Manickam, and Mohammed Anbar (Eds.). Springer, Singapore, 505–514.

[26] Hamid Reza Mahdiani, Sied Mehdi Fakhraie, and Caro Lucas. 2012. Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors. *IEEE Trans. Neural Netw. Learn. Syst.* 23, 8 (2012), 1215–1228. https://doi.org/10.1109/TNNLS.2012.2199517

[27] Shayan Moini, Shanquan Tian, Daniel Holcomb, Jakub Szefer, and Russell Tessier. 2021. Power side-channel attacks on BNN accelerators in remote FPGAs. *IEEE J. Emerg. Select. Top. Circ. Syst.* 11, 2 (2021), 357–370. https://doi.org/10.1109/JETCAS.2021.3074608

[28] Preeti Nagrath, Rachna Jain, Agam Madan, Rohan Arora, Piyush Kataria, and Jude Hemanth. 2021. SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2. *Sustain. Cities Soc.* 66 (2021), 102692. https://doi.org/10.1016/j.scs.2020.102692

[29] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2019. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV'19)*.

[30] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2020. TBT: Targeted neural network attack with bit Trojan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'20)*.

[31] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC'18)*. 1–6. https://doi.org/10.1109/DAC.2018.8465834

[32] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2019. An efficient bit-flip resilience optimization method for deep neural networks. In *Proceedings of the Design, Automation and Test in Europe (DATE'19)*. 1507–1512. https://doi.org/10.23919/DATE.2019.8714885

[33] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet Things J.* 3, 5 (2016), 637–646.

[34] Donghoon Shin, Hyun-geun Kim, Kang-moon Park, and Kyongsu Yi. 2020. Development of deep learning-based human-centered threat assessment for application to automated driving vehicle. *Appl. Sci.* 10, 1 (2020). https://doi.org/10.3390/app10010253

[35] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. Retrieved from https://arXiv:1409.1556.

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'15)*.

[37] Shanquan Tian, Ilias Giechaskiel, Wenjie Xiong, and Jakub Szefer. 2021. Cloud FPGA cartography using PCIe contention. In *Proceedings of the IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'21)*. 224–232. https://doi.org/10.1109/FCCM51124.2021.00035

[38] Xiuxiu Wang, Yipei Niu, Fangming Liu, and Zichen Xu. 2022. When FPGA meets cloud: A first look at performance. *IEEE Trans. Cloud Comput.* 10, 2 (2022), 1344–1357. https://doi.org/10.1109/TCC.2020.2992548

[39] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. 2020. Leaky DNN: Stealing deep-learning model secret with GPU context-switching side-channel. In *Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'20)*. 125–137. https://doi.org/10.1109/DSN48063.2020.00031

[40] Zane Weissman, Thore Tiemann, Daniel Moghimi, Evan Custodio, Thomas Eisenbarth, and Berk Sunar. 2019. Jackhammer: Efficient RowHammer on heterogeneous FPGA-CPU platforms. Retrieved from https://arXiv:1912.11523.

[41] Chunpeng Wu, Wei Fan, Yuan He, Jun Sun, and Satoshi Naoi. 2014. Handwritten character recognition by alternately trained relaxation convolutional neural network. In *Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition*. 291–296. https://doi.org/10.1109/ICFHR.2014.56

[42] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. 2020. DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security'20)*. USENIX Association, 1463–1480. https://www.usenix.org/conference/usenixsecurity20/presentation/yao.

[43] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2022. Enabling cost-effective, SLO-aware machine learning inference serving on public cloud. *IEEE Trans. Cloud Comput.* 10, 3 (2022), 1765–1779. https://doi.org/10.1109/TCC.2020.3006751

[44] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. 2021. Stealing neural network structure through remote FPGA side-channel analysis. *IEEE Trans. Info. Forens. Secur.* 16 (2021), 4377–4388. https://doi.org/10.1109/TIFS.2021.3106169

[45] Mark Zhao and G. Edward Suh. 2018. FPGA-based remote power side-channel attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'18)*. 229–244. https://doi.org/10.1109/SP.2018.00049

[46] Pu Zhao, Siyue Wang, Cheng Gongye, Yanzhi Wang, Yunsi Fei, and Xue Lin. 2019. Fault sneaking attack: A stealthy framework for misleading deep neural networks. In *Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC'19)*. 1–6.

Received 24 June 2022; revised 27 October 2022; accepted 15 December 2022