information

Article

Attacking Deep Learning AI Hardware with Universal Adversarial Perturbation

Mehdi Sadi [1,*], Bashir Mohammad Sabquat Bahar Talukder [2], Kaniz Mishty [1] and Md Tauhidur Rahman [2,*]

[1] Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849, USA; kzm0114@auburn.edu
[2] Department of Electrical and Computer Engineering, Florida International University, Miami, FL 33199, USA; bbaha007@fiu.edu
* Correspondence: mehdi.sadi@auburn.edu (M.S.); mdtrahma@fiu.edu (M.T.R.)

Abstract: Universal adversarial perturbations are image-agnostic and model-independent noise that, when added to any image, can mislead the trained deep convolutional neural networks into the wrong prediction. Since these universal adversarial perturbations can seriously jeopardize the security and integrity of practical deep learning applications, the existing techniques use additional neural networks to detect the existence of these noises at the input image source. In this paper, we demonstrate an attack strategy that, when activated by rogue means (e.g., malware, trojan), can bypass these existing countermeasures by augmenting the adversarial noise at the AI hardware accelerator stage. We demonstrate the accelerator-level universal adversarial noise attack on several deep learning models using co-simulation of the software kernel of the Conv2D function and the Verilog RTL model of the hardware under the FuseSoC environment.

Keywords: universal adversarial perturbations; deep learning accelerator; AI hardware security

MSC: 2020;68

Citation: Sadi, M.; Talukder, B.M.S.B.; Mishty, K.; Rahman. M.T. Attacking Deep Learning AI Hardware with Universal Adversarial Perturbation. *Information* **2023**, *14*, 516. https://doi.org/10.3390/info14090516

Academic Editor: Paolo Maistri

Received: 24 July 2023
Revised: 4 September 2023
Accepted: 6 September 2023
Published: 19 September 2023

1. Introduction

The last decade has been a conspicuous success in the history of AI. Using deep neural networks (DNN), AI has superseded human intelligence in many applications. For example, Deepmind's AlphaGo has defeated the world's best human Go player, AlphaFold can successfully predict the 3D protein fold structure, which had been a challenge for the past 50 years [1], and, since 2015, ImageNet classifier models have been outperforming humans in object recognition [2]. The rapid advancement in model building and their promising accuracy in particular tasks have led DNN models to be frequently deployed in a wide range of applications, including many safety-critical areas such as biometric security, autonomous vehicles, cyber security, health, and financial planning. To withstand the overarching AI-based applications, which require massive computations, the demand for specially optimized hardware for these extremely power-hungry and data-driven computations has also surged both in the data centers (for training) and edge devices (for inference) [3,4]. As these AI accelerator devices are being used in several safety-critical applications, ensuring their security and integrity is essential.

While AI and deep learning have become pervasive in all aspects of our lives, we should not disregard the fact that the DNN models are highly vulnerable to adversarial examples [5]. An adversarial example is an example that is intentionally created to be misclassified by a DNN model by adding adversarial perturbation to the input data [5]. The addition of this adversarial perturbation to clean inputs makes the inputs falsely classified by state-of-the-art DNN models while retaining credibility with humans. Adversarial samples are obtained by adding imperceptibly small perturbations to a correctly classified input image so that it is no longer classified correctly. The adversarial samples expose

fundamental blind spots in the DNN training algorithms. The adversarial examples represent low-probability (high-dimensional) "pockets" in the input space, which are hard to efficiently find by simply randomly sampling the input around a given example [5,6]. By using a simple optimization procedure, adversarial examples can be found. The *generalization* (i.e., adversarial perturbation created for a model trained with one data distribution is also harmful to the models that are trained over different sets of data distribution) and *transferability* (i.e., the adversarial examples computed for one model are also valid for every other model with completely different architectures, hyperparameters, and training data) properties of adversarial examples have doubly contributed to this stealthy nature of DNN models [5–8].

The existence of the above-described features of DNN provides enough opportunity to an attacker to intrude on an AI model to inflict massive damage in his intended domain. Depending on the extent of information (such as model parameters, input, or output) available to the adversary and in which phase the attack has been launched, various types of attacks have emerged in the last several years. In a poisoning attack, a malicious model is developed during the training phase either by training the model on adversarial data distribution, altering the training procedure, or manipulating the model's weights or architecture [9,10]. In contrast, in an evasion attack, an adversary discovers malicious inputs on which the model will make unexpected errors [9]. When the adversary has access to all sorts of necessary information about the model, such as model architecture, model parameters, and weights, the attack scenario is termed a white-box attack [11]. Contrarily, in a black-box attack, an attacker cannot access any model information other than the model's response to the chosen input [11]. Upon its discovery, it has been shown that the adversarial examples can be exploited to successfully attack all kinds of deep learning applications [12]. However, all these attacks are software-based and mainly focus on devising innovative techniques to generate adversarial examples. It is assumed that the hardware where the DNN models are computed (training and inference) is always reliable and trustworthy. Unfortunately, this assumption is no longer valid in the current semiconductor industry, and security vulnerabilities in the hardware, as well as software malware, call for serious attention from the research community [13,14].

AI/deep learning hardware can be compromised from methods based on both hardware (e.g., trojans) and software (e.g., malware). Because of ASICs' complex and intricate structure, semiconductor industries involve design outsourcing and globalization of fabrication. In most cases, the designed GDSIIs travel overseas, from designer to foundry, before they are finally deployed in the target device. This whole supply chain flow has become one of the most prominent causes of the infiltration of untrusted hardware in the semiconductor industry [13]. Anyone with an evil intent engaged in this flow is capable of any malicious alteration of the target product. Because of this reason, hardware security has received attention in the past decade. However, very few works have been completed regarding an AI-specialized hardware platform. From the software perspective, malicious code in the form of malware [15–17] can compromise deep learning/AI computing systems. Although malware protection techniques and computer system security have significantly improved over the past decade, attackers are consistently coming up with new evasive and sophisticated strategies to breach the software- and hardware-level malware detection techniques to execute malicious functions [15–17]. Preventing attackers from installing malicious programs to gain privileged access to the system is practically impossible as they can exploit various techniques, such as web browser vulnerabilities, social engineering (e.g., phishing), email attachments, flash drives, etc. Symantec reported that 246,002,762 new malware variants emerged in 2018 [15]. Adversarial noise can be injected into deep learning AI hardware by compromising its operating system and execution software with various malware attacks similar to Stuxnet [18].

In this work, for our attack strategy, we exploit the recently discovered phenomenon that there exists a *universal adversarial perturbation (UAP)* noise that, when added to the inputs of the DNN model, can severely compromise their prediction accuracy [8,19]. The

following scenario can be pictured as an example. An inference-only device is used for the inference tasks of a pre-trained AI model in an autonomous vehicle. The task of the model is to provide an appropriate decision about surrounding objects and road signs to the vehicle. The DNN model is authentic. However, the device that is performing the computation has a severe security flaw: it has a *universal adversarial noise* image stored in its memory, which can be augmented to the input image captured by the vehicle's camera and is capable of fooling almost all DNN models irrespective of their architecture and training data distribution. The model will then decide on the perturbed image and provide a disastrous decision for the car based on its misclassified output. The existing fault-injection- and bit-flip-based attacks [20–24] require a fair amount of knowledge about network architectures and parameters to materialize an attack by careful manipulation of the network. These constraints make these attacks hard to implement in real time. In contrast, the *UAP* attacks are more generic and, hence, more destructive.

In this paper, we present a novel accelerator-based DNN attack that exploits the *UAP* noise, which does not need any information about the model's architecture and parameters. The attacker only needs to have the scope of planting malware or software/hardware trojans. The key contributions and highlights of this paper are

- To the best of our knowledge, this work, for the first time, proposes and demonstrates an accelerator-based DNN attack that requires little to no knowledge about the target DNN model by exploiting *universal adversarial perturbation (UAP)*. The proposed attack is sneaky enough to obscure its existence yet powerful enough to cause massive damage.
- We propose a novel technique to interleave the UAP noise with the actual image to mislead the trained DNN model when the attack is activated with malware or software/hardware trojans. Since our technique avoids the usual methods of adversarial noise injection at the sensor or camera level and directly injects at the accelerator hardware stage, it is more stealthy.
- We provide a detailed comparative analysis of the complexity, stealth, and implementation challenges of the existing Trojan, fault injection, bit-flip-based, and proposed UAP-based attacks on the AI/deep learning hardware.

The rest of the paper is organized as follows. Section 2 presents the background on DNNs, accelerators, and adversarial noise attacks. Section 3 presents the threat model of the UAP attack. Section 4 presents the details of the UAP interleaving method. The stealth of the proposed UAP attack is discussed in Section 5. The experimental results are presented in Section 6 and related work in Section 7.

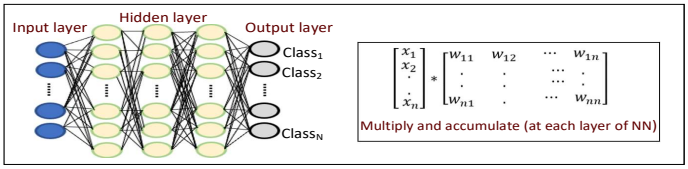
2. Background

2.1. AI/Deep Learning Neural Networks

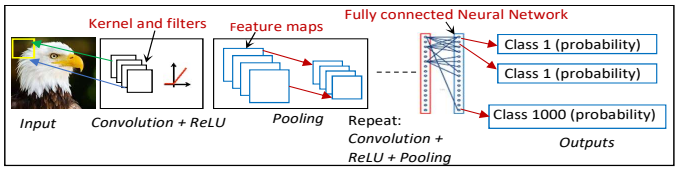
The backpropagation-based NN and CNN are the essence of AI/deep learning algorithms. As shown in Figure 1a, a deep NN consists of an input layer, followed by several hidden layers and a final output layer. Depending on the data size, complexity of training, dropout, and pruning rate, some layers in the NN are fully connected, and others are sparsely connected [2]. The connection strengths between the adjacent layers are represented by a weight matrix W, and the matrix parameters w_i are learned by the backpropagation-based learning equation, $\Delta w_i = -\alpha \times \frac{\partial Error}{\partial w_i}$, where α is the learning rate, and $Error$ is the prediction error. During the forward pass of training and inference phases, the output activation of a layer, X_o, is obtained by multiplying the input activation vector with the weight matrix followed by the addition of a bias term, and finally passing the result through a non-linear function, such as ReLU, $X_o = ReLu(W \times X_i + b)$.

Due to their higher accuracy, deep CNNs have become the standard for image and pattern recognition [2]. The operation of a deep CNN is briefly shown in Figure 1b. During training and inference, each image or pattern is convolved successively with a set of filters where each filter has a set of kernels. After ReLU activation and pooling, the convolution operation is repeated with a new set of filters. Finally, before the output stage, fully

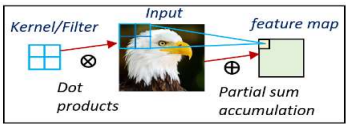
connected NNs are used. The convolution operation is shown in Figure 1c and consists of dot products between the input feature maps and filter weights (h), mathematically, $f_{out}(m,n) = \sum_j \sum_k h(j,k) f_{in}(m-j, n-k)$.



(a) Deep Neural Network (NN)



(b) Deep Convolutional Neural Network (CNN)

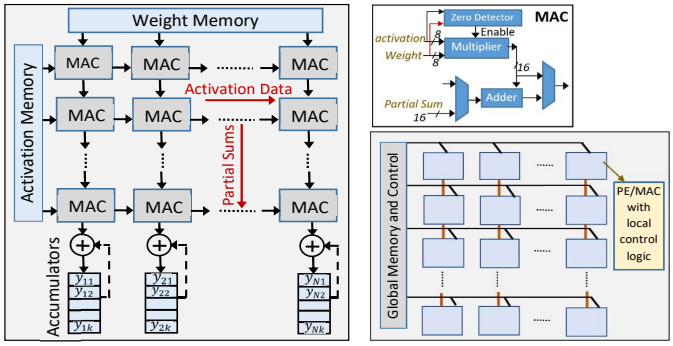


(c) Multiply and accumulate (MAC) in convolution operation

Figure 1. (**a**) Deep NN; (**b**) deep CNN; (**c**) convolution in CNN.

2.2. AI/Deep Learning Accelerator Architecture

Since the computations in deep NN/CNN are mostly dominated by Multiply and Accumulate (MAC) operations, the AI accelerators are primarily occupied with arrays of Processing Elements (PE) optimized for fast MAC function [2]. As shown in Figure 2, the accelerator architectures can be categorized into two domains: (i) tightly coupled 2D systolic arrays (e.g., Google's TPU) [2,3] and (ii) loosely coupled spatial arrays with independent PEs interconnected with NoC/mesh and using SIMD architecture [2].



(a) (b)

Figure 2. AI accelerator with PE/MAC arrays. (**a**) Systolic architecture; (**b**) SIMD architecture.

2.3. Adversarial Perturbation to Mislead Trained AI Models

The well-trained state-of-the-art deep learning models can be fooled to misclassify inputs by augmenting the input images with adversarial patterns. Adversarial samples are created by adding imperceptibly small and non-random perturbations to the correctly predicted original input images to cause the trained deep learning models to classify the input images falsely [5–8]. Because of the small magnitude of the perturbation, the adversarial samples are often imperceptible to the human eye and correctly classified by a human observer but fail at the trained deep learning model level. The adversarial samples are distinct from randomly distorted samples. In most cases, the randomly distorted image samples—where the magnitude of the random noise is not too large compared to the actual

image—are classified correctly by the state-of-the-art deep learning networks. However, the adversarial samples are almost always misclassified [5–8]. Two important properties of adversarial perturbations are as follows [6].

- *Generalization Property:* Deep learning models demonstrate consistency in misclassifying adversarial samples. Adversarial images created for one deep learning model are often erroneously classified by other models with different hyperparameters (i.e., different model architecture, number of layers, initial weights, regularization, etc.) and a different subset of training data [5–8].
- *Transferability Property:* The adversarial samples are robust and demonstrate transferability property [5–8]. Adversarial samples crafted to mislead a specific deep learning model are also effective in misclassifying other deep learning models, even if their architectures greatly differ.

These two properties make adversarial attacks more pervasive and stealthy. The adversarial patterns can be maliciously augmented with the input data at the AI hardware level and cause the trained deep learning model to fail.

Adversarial Sample Crafting

There are two classes of methods for generating adversarial samples.

- *Image-based Adversarial Perturbation:* In this method [5–7], adversarial samples are created on a per-image basis by adding carefully crafted noise to the original image along the gradient directions. Once a particular image is augmented with this noise, it can show its adversarial efficacy across different neural network models. However, to achieve a successful adversarial goal, the perturbation needs to be generated separately for each image [5,6].

Using the Fast Gradient Sign Method (FGSM), adversarial samples can be created in this approach [6,7]. The adversarial input $\overrightarrow{x^*}$ can be generated from the input pattern $\overrightarrow{x}$ by adding a perturbation, i.e., $\overrightarrow{x^*} = \overrightarrow{x} + \overrightarrow{\eta}$, The perturbation is obtained with FGSM using the equation $\overrightarrow{\eta} = \epsilon * sign(\nabla_{\overrightarrow{x}} J(\theta, \overrightarrow{x}, y))$, where θ is the model parameter, $\overrightarrow{x}$ is input to the model, y is the target, and $J(\theta, x, y)$ is the cost function. The gradient is calculated using backpropagation [5,6].

- *Universal Adversarial Perturbations:* In the second method [8], universal adversarial perturbations are generated based on the input (i.e., image) data distribution rather than individual images. In addition to being network-agnostic (i.e., transferable among different state-of-the-art deep learning models), these universal perturbations are image-agnostic and maintain their adversarial efficacy across different images belonging to the same distribution (e.g., ImageNet dataset). For a classification function f that outputs a predicted label $f(x)$ for each image $x \in \mathbb{R}^d$, a perturbation vector $v \in \mathbb{R}^d$ is the universal adversarial perturbation vector that misleads the classifier f on almost all data points sampled from the distribution of images in $\mathbb{R}^d$ such that $f(x + v) \neq f(x)$ for most x in the distribution [8]. The algorithm to find v proceeds iteratively over a set of images sampled from the distribution and gradually builds the universal perturbations. Due to their small magnitude, the perturbations are hard to detect and do not significantly alter the data distributions. The universal perturbations were shown to have generalization and transferability properties across different architectures on the ImageNet dataset. This implies that, to fool a new image on an unknown deep neural network, a simple augmentation of a universal perturbation generated on AlexNet/VGG16 architecture is highly likely to misclassify the image.

2.4. Adversarial Attack Strategy during Deep Learning Inference

The adversarial attacks on trained deep learning models can be broadly classified as white-box and black-box types. In the white-box attack scenario, for a target deep

learning model, the adversary has access to all the elements of the training procedure, such as the model architecture, training data, training algorithm, weights, etc. With this information, in a white-box attack scenario, the adversary can use arbitrary methods to create the adversarial samples [7]. In a stronger black-box attack scenario, the adversary only has access to the deep learning model's input–output relationships. In the black-box attack in [11], it was demonstrated that adversarial attacks could be manifested by developing a surrogate model of the target DNN under attack by knowing only the output class information of the model for various inputs (i.e., Oracle query [7]). In this black-box attack mode—without truly knowing the architecture of the model being attacked—using the surrogate model, the attacker can generate adversarial samples. The transferability property (i.e., adversarial sample created on one deep learning model is also effective on another) of adversarial perturbation ensures that the samples will be effective on the actual DNN under attack [7].

For the case of universal adversarial-perturbation-based attacks [8], the attack strategy becomes even more straightforward. Because of the image-agnostic universal nature of the perturbation, the attacker can create the universal adversarial perturbations by only knowing that the AI/deep learning model is completing image recognition/classification tasks. For example, universal adversarial perturbations created based on the large ImageNet dataset demonstrate its adversarial effectiveness across various state-of-the-art image classification deep learning models by virtue of the *generalization* and *transferability* properties of adversarial samples [8].

To counteract adversarial attacks of the deep learning models, adversarial training was proposed where adversarial samples were used during the training procedure. However, in [7], it was demonstrated that the adversarially trained models remained vulnerable to multistep and elaborate black-box attacks where perturbations created on undefended models were transferred to the adversarially trained models.

3. Threat Model

Attacker Knowledge: The attack model is BlackBox [11] as it is not required to know the model parameters, training data, etc. The attacker only needs to know the deep learning model's task category (e.g., image classifier) and use the appropriate universal adversarial perturbation. In our threat model, to materialize the attack, it is not required for the trained model to have back doors [11,25,26]. These make the attack more stealthy.

Adversarial Goal: Once the attack mode is activated with malicious means (e.g., malware), the adversarial noise is injected into the AI hardware, and the attacker gains the ability to fool the pre-trained AI/deep learning model into misclassifying the input samples during inference.

Attack Strategy: The strategy for attacking AI hardware accelerator with universal adversarial perturbation is shown in Table 1. To implement this attack in the AI accelerator hardware, with heightened privilege (root access on Linux, or admin access in Windows), the adversary can access and modify the target program's address space through the */proc/[PID]/map* and through .DLL (Dynamic Link Library) injection in Unix and Windows operating systems, respectively [27–29]. Specific types of malware, such as trojan, rootkit, and fileless, can covertly alter the system library to accomplish malicious program execution [15,30]. Trojan-type malware such as Emotet uses functionality that helps evade detection by some anti-malware products [15]. Fileless malware (e.g., the Astaroth malware) are unique as they are capable of altering files related to the operating system without installing any files of their own [30]. As the targets of the fileless malware are part of the operating system, they often appear as legitimate files to the antivirus/malware tools, and this makes the attacks stealthy [30]. One of the most dangerous malware, Rootkit [31], provides privileged access to the computing system to the attacker. Once the rootkit is installed in the system, the adversary can maintain privileged access and full control over the system without detection, as it can disable malware detectors. A rootkit can target

specific applications and partially modify their execution behavior by injecting its own code. Another approach to malicious code/function injection is to remap memory between processes with a malicious kernel module, which has proved effective in many well-known attacks like Stuxnet [18].

Table 1. Strategy for attacking AI hardware accelerators with universal adversarial perturbation.

Attack Type	Attack Strategy (Details in Section 4)
Malicious Noise Interleaving and Convolution (MNIC)	Universal adversarial noise is interleaved with the original image, and the filter rows are repeated before the first convolution operation (see Figure 3). Malicious modification (e.g., with malware) of the inputs of the 'Conv2D' function can accomplish this task.

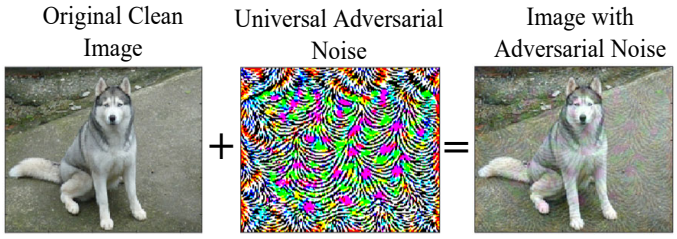


Figure 3. Universal adversarial perturbation added with a clean image to deceive the trained deep learning model.

4. Accelerator-Level Addition of Adversarial Perturbation

To fool the trained AI/deep learning models, the adversarial perturbation noise needs to be added to the input image before it is processed with convolution filters. In a regular setting, to superimpose the adversarial patterns on the image, modifications at the camera or the input image source itself are required before the AI accelerator processes it. Since this input alteration adversarial attack is well-known, the image capture sensors and digitization modules are protected with security measures to prevent any adversarial noise additions. In our hardware-level attack, we assume the image sources are protected and clean, and in our attack model, the universal adversarial perturbations are added to the clean images by the attacker at the SoC level by exploiting the internal memory system and software with malware/trojan. A simplified block diagram of an AI accelerator hardware system is shown in Figure 4. The host CPU receives the clean input images through the input port or sensors and then sends those to the accelerator hardware through the DRAM for efficient and fast processing. In our attack model, the universal adversarial pattern is stored on the SoC at the ROM or some other on-chip non-volatile memory. The adversarial pattern can also be transferred to the system externally as malware. Under malicious attack, this adversarial pattern is added with each input image before it is processed at the AI accelerator hardware. Because of the transferability and generalization properties of universal adversarial perturbations (i.e., discussed in Section 2.3), a single well-crafted pattern can fool many images with high efficiency [8].

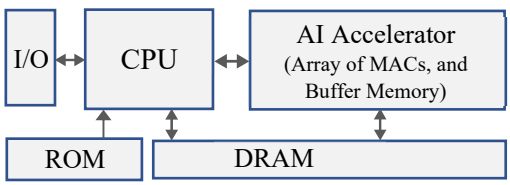


Figure 4. AI hardware accelerator architecture.

Directly adding the adversarial noise with the input image using the CPU's ALU will require many additions, which may raise the security alarm, indicating a malicious

attack. For example, for typical RGB images of 3 channels (e.g., from ImageNet dataset), $224 \times 224 \times 3$ addition operations will be required at the ALU to add the adversarial noise with the input before sending it to the AI accelerator. As a result, in our attack strategy, we avoid directly adding the stored universal adversarial perturbation with the image at the host CPU. We utilize the additive property (i.e., $f * (a + b) = f * a + f * b$, where $*$ is the convolution operator) of convolution operation to create the same effect as convolving the filter with the image perturbed with adversarial noise without directly adding the noise to the image. This operation is mathematically explained with the illustration of Figure 5. In Figure 5a, Y_{jk} is the noise-added image pixels, i.e., $Y_{jk} = I_{jk} + n_{jk}$, and I_{jk} and n_{jk} are clean image and noise, respectively, for pixel location jk. The output of convolution with filter F is O. In Figure 5b, instead of directly adding the noise pixel n_{jk} with corresponding clean image pixels I_{jk}, the noise pixel rows are interleaved with the image pixels. To produce the exact same outputs of Figure 5b, O_{mn}, two other adjustments are necessary in Figure 5b: (i) duplication of filter rows and (ii) doubling the vertical stride of convolution, as shown in Figure 5b. With this modification, we can produce the same effect of convolving the filter with noise-added inputs without ever explicitly adding the noise. In Figure 6, an example is shown where universal adversarial noise patterns are interleaved across rows with an image from the ImageNet dataset.

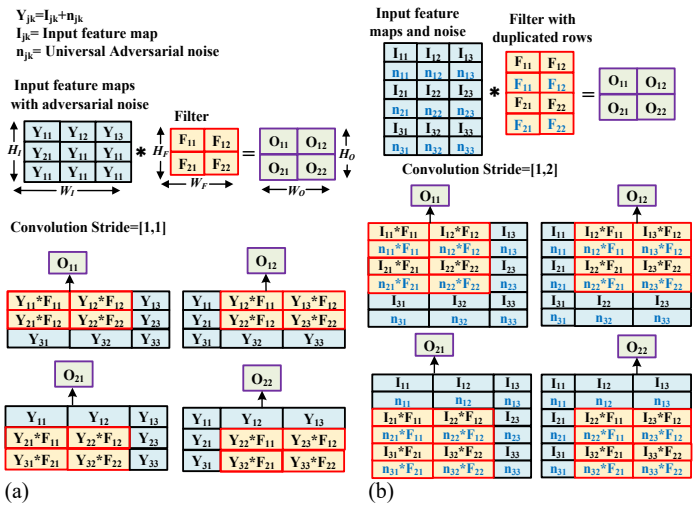


Figure 5. Using the additive property of convolution, the same output feature map can be generated by noise interleaving without directly adding noise to the input. (**a**) Convolution with direct adversarial noise addition with input; (**b**) noise interleaved with input, filter duplicated, and vertical stride doubled to accomplish the same task.

We need to perform this (i) noise-interleaved and (ii) repeated filter-row-based convolution only for the first layer of the deep learning model, and the rest of the layers follow the regular convolution approach. Since we are doubling the filter rows in this approach, the number of MAC operations will also double only for the first layer of the model. However, in contrast to the ALU of the CPU, the AI accelerator is equipped with a large number of MAC modules (e.g., Google TPU has 65K MACs [3]), and, hence, the extra MAC operations will complete very fast without raising any security alarm. Moreover, because of zero-skip (i.e., if any of the MAC inputs are zero, the operation is skipped), and the presence of a variable number of zeros in digitized images and quantized filter weights, the number of MAC operations is not deterministic (i.e., fixed) for clean images. As a result of the presence of a large number of MAC arrays in AI accelerators and zero-skipping in modern accelerators, the extra MAC operations (i.e., only in the first layer) of our noise-interleaved convolution will become masked to the extent that they do not raise any security alarm.

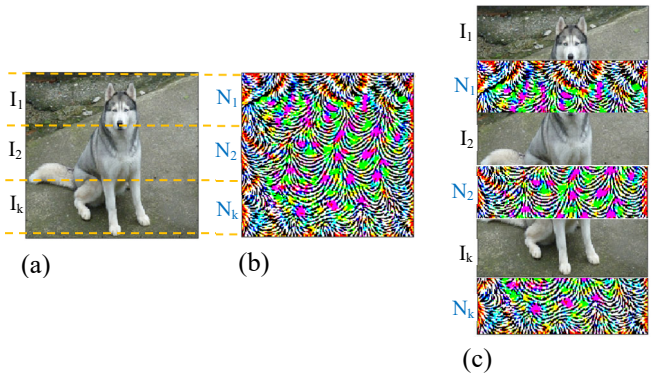


Figure 6. (**a**) Clean input image; (**b**) universal adversarial perturbation; (**c**) adversarial perturbation interleaved with the input image.

The AI accelerator loads the input image in its global buffer memory from the DRAM [2,3]. In the regular scenario, the input image rows are placed adjacently in the Memory as shown in Figure 7a. In our proposed hardware-level attack scheme, the rows of universal adversarial perturbation are placed adjacent to the image rows in the Memory as shown in Figure 7b. This can be achieved by malware or software/hardware trojans. As a result, during execution at the accelerator, the noise data are interleaved with the clean image data, and ensuing convolution in the accelerator, in effect, convolves the filter with images corrupted with universal adversarial perturbation as graphically shown in Figure 5b.

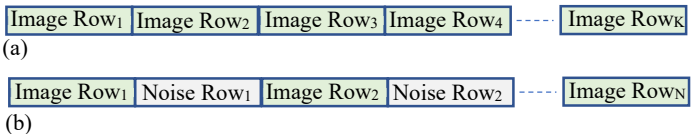


Figure 7. Input image data positioning in memory: (**a**) regular scenario; (**b**) under adversarial attack, noise data are interleaved.

The hardware-level malicious addition of the adversarial perturbation attack can be initiated with malware or software-controlled trojans [32–34]. Additionally, the attacker can utilize the unused instruction in the instruction set to manifest the adversarial noise injection attack.

5. Stealth against Adversarial Detection Techniques

Our attack strategy can evade conventional protections against adversarial attacks in AI/deep learning systems, as discussed in detail in the following.

5.1. SGX Independence

Intel SGX [35] can only protect the data and code segments stored in the *Enclave*. However, in our threat model, (1) the adversarial noise data (about 74KB) and (2) the malicious "Conv2D" function (few lines of code) are generic and already known to the attacker. As a result, the attacker adds these data and code segments to the original version even before they are transferred to the protected *Enclave*. In our threat model, the attacker does not need to modify data in *Enclave*; thus, the attack scenario is independent of SGX.

5.2. Rowhammer Independence

In the conventional bit-flip attack (i.e., where the attacker needs to modify the trained model's weights) [21], rowhammer-based weight bit flipping in DRAM is necessary to materialize the attack. With rowhammer-protected DRAM circuit design and Intel SGX, the DRAM-based bit-flip attacks can be thwarted. Our threat model is non-invasive to the DNN model as we do not need to tamper with the original trained model parameters.

5.3. Bypassing Conventional Countermeasures against Adversarial Noise

In conventional attack models, the adversarial noise is generally added at the image source; hence, protective measures are taken accordingly. However, in our attack strategy, adversarial noise is added internally at the AI accelerator level. As a result, it can bypass the state-of-the-art adversarial detection techniques that assume adversarial noise is added at the image source, making the attack stealthier. The SentiNet method presented in [26] detects adversarial attacks, such as trojan triggers, backdoors, poisoning, and adversarial patches, on test images. SentiNet first identifies the salient regions of an image that highly influence its class labels and therefore are potential targets of adversarial noise addition. To identify if the test image is compromised, these salient regions are superimposed on benign golden samples and tested if they are misclassified upon overlaying. High misclassification rates indicate the presence of adversarial noise or patches. The STRIP technique proposed in [25] detects trojan images on the concept that, if an image is backdoored with adversarial patches, it will always classify into a certain adversary-chosen class. The STRIP detection algorithm adds perturbations to the test image and identifies if its class label changes and a static class label indicates that the image was trojaned with an adversarial patch. However, to implement these approaches at run time, additional AI accelerator hardware is needed to execute these detection models in parallel to the original deep learning model, and this incurs extra energy and latency. Moreover, these detection techniques are developed for scenarios where the adversarial noise or trojan is localized to a patch, in contrast to our threat model, where low-magnitude universal noise spans across the image.

In [36], a defense mechanism against UAP has been presented. First, a Perturbation Rectifying Network (PRN) is trained and then used as the 'pre-processing' layer of the targeted model. Next, a perturbation detector is separately trained as a binary classifier to extract discriminative features from the difference between the inputs and outputs of the PRN. However, this proposed method requires extra pre-trained models to be added before the target model and also assumes a conventional scenario where the images are compromised with UAP before they are processed at the accelerator. As in our proposed attack, the UAP is interleaved with the images at the hardware accelerator stage; the detection techniques of [36] can be evaded and the attack becomes stealth.

5.4. Non-Deterministic Execution

In our approach, because of duplicating the filter rows of the first layer, the number of MAC operations in the first layer of the model can double. However, this may not raise any security alarm because, in modern AI accelerators, zero-skipping is implemented [2] where, if any of the ifmap or filter input is zero, MAC operation is skipped. Because of image digitization and quantization before feeding to the accelerator, there are several zeros in the input image pattern and this number varies from image to image. Moreover, there are also zeros present in the filter. As a result, the number of MAC operations per image is not deterministic, and, hence, the extra MAC required in our adversarial attack will vary from image to image without showing any constant pattern that can raise the alarm.

6. Experimental Results

To generate the universal adversarial perturbations, we used the tools available from [8]. The normalized maximum magnitude of the adversarial perturbation was kept within 5% of the normalized maximum magnitude of the images. For our analysis, we used the standard 50K validation image samples from the ImageNet database. After adding the universal adversarial perturbations with these images, we used the pre-trained version of several widely used deep learning models and analyzed the fooling rate of adversarial noise with PyTorch [37]. In accordance with [8], we define the fooling rate as the percentage of image samples that altered their labels under adversarial noise compared to clean images (i.e., irrespective of ground truth). We used the same (i.e., a single pattern of 224 $\times$ 224 pixels with three channels) universal adversarial noise (i.e., *transferability* property as discussed in Section 2) for our experiments with AlexNet, ResNet-50, VGG-16, and GoogleNet deep

learning models. The fooling rates are shown in Table 2. We can observe that more than 80% of validation image samples altered their labels when adversarial noise was added.

Table 2. Adversarial fooling rate.

Network	AlexNet	VGG-16	ResNet-50	GoogleNet
Adversarial Fooling Rate (%)	90.8	88.9	84.2	85.3

The normalized Top-1 and Top-5 accuracy changes for ImageNet data under the universal adversarial perturbation are shown in Figure 8. Additionally, to demonstrate the efficacy of universal adversarial noise compared to random noise, we also ran two sets of additional experiments where (i) the clean images were augmented with low-magnitude random noise (i.e., noise magnitude within 5% of the original image, similar to the case of the adversarial perturbation) and (ii) high-magnitude (i.e., maximum allowed noise magnitude same as the image magnitude) random noise added with the clean images. The results for AlexNet, VGG-16, ResNet-50, and GoogleNet are shown in Figure 8. From these experiments, it can be seen in Figure 8 that random noise with an amplitude similar to adversarial perturbation only slightly impacts the accuracy, implying that the sate-of-the-art AI/deep learning models are highly immune to small random noise at the input. However, carefully crafted universal adversarial noise of the same low magnitude range (i.e., 5% of the original image) can drastically compromise the fidelity and accuracy of the models. To achieve similar accuracy degradation effects, the random noise magnitude must be large (i.e., in the same order as the image), as shown in Figure 8. However, compared to the low-magnitude adversarial noise, augmenting high-magnitude random noise at the AI accelerator level is a complex task because of the larger bit width and memory size requirements (i.e., 4 bits per pixel for adversarial noise versus 8 bits for random noise), and this raises practical implementatrion challenges from a hardware-based attack perspective.

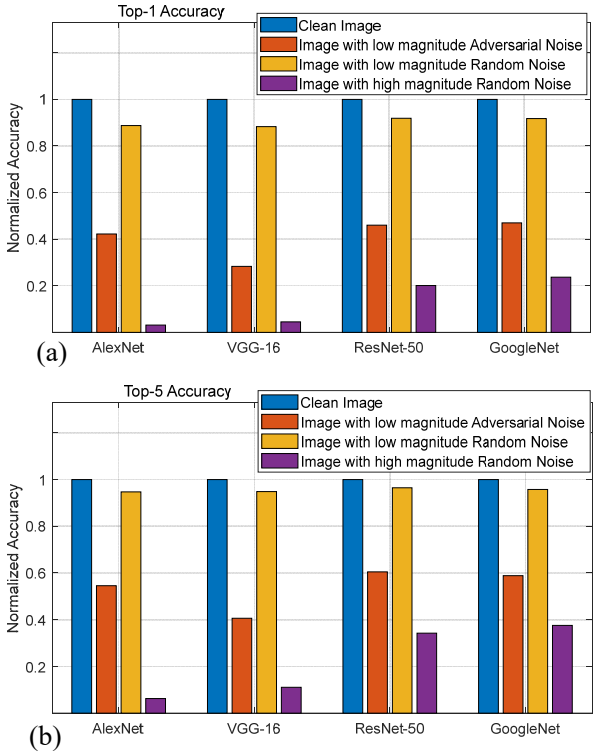


Figure 8. Normalized prediction accuracy changes for (i) clean, (ii) adversarially perturbed, (iii) low-magnitude random-noise-augmented, and (iv) high-magnitude random-noise-added images for ImageNet benchmark. (**a**) Top-1 accuracy; (**b**) Top-5 accuracy.

With a large magnitude noise, the amount of non-zero bits in the digitized pattern will be significant and comparable to the image itself. As a result, hardware accelerator-level malicious addition of the noise pattern can raise security alarms. On the contrary, universal adversarial patterns can induce significant accuracy degradation; however, because of their much lower magnitude compared to the actual image, they can be injected more secretively. For example, for images from the imageNet dataset with $224 \times 224 \times 3$ pixels and 8 bits per pixel, to achieve the same level of image misclassification as the adversarial noise with high-magnitude (e.g., noise magnitude at least 50% of image magnitude) random noise, 90% more noise bits are required.

Moreover, because of the lower magnitude of the adversarial noise, the pattern mostly consists of zeros, and the low number of non-zero bits only occurs in the lower-level bits (e.g., LSBs) of the digitized adversarial pattern. For ImageNet benchmark images with 8 bits per digitied pixel of the clean image, the pixels of the low-magnitude (i.e., noise magnitude limited within 5% of the original image magnitude) universal adversarial perturbation can be digitized with a maximum of 4 bits. Also, the low-bit-width adversarial noise can be compressed using conventional sparse weight compression techniques of quantized AI models [2] and easily hidden in the AI hardware for later malicious deployment in a stealthy manner.

To demonstrate the hardware-level adversarial attack on AI/deep learning models based on the interleaved augmentation of universal adversarial perturbation with images (i.e., described in Section 3) on a SoC platform, we used the OpenRISC-based MOR1K soft CPU core with FuseSoC platform [38]. The FuseSoC platform allows writing high-level C and assembly code programs that can be compiled and executed on the MOR1K CPU with other added hardware modules. First, we present the C programming code example of the Conv2D function (i.e., the convolution function used in convolutional neural networks) to demonstrate how input image and adversarial noise data interleaving, filter row duplication, and convolution stride modifications can be accomplished. In Figure 9a, the C code for a regular Conv2D task with $Stride = 1$ is shown. Filter row duplication and interleaving of the adversarial perturbation with an image before feeding it to the Conv2D task is shown in the C code of Figure 9b. After running the MOR1K toolchain [38] under the FuseSoC environment, the C code is converted to equivalent machine code as shown in the snippets in the bottom segments of Figure 9, where it can be seen how the *Stride* parameter can be altered in the machine code.

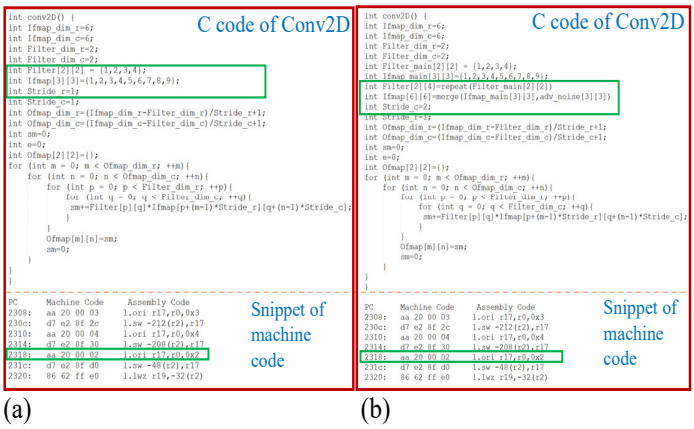


Figure 9. An example of Conv2D code. Under attack, the machine code can be compromised to inject universal adversarial perturbation in the images. (**a**) Regular code. (**b**) Merging adversarial perturbation by image interleaving and altering the stride.

The attack can be manifested both at the high-level C code kernel stage and at the compiled machine code stage. Based on the illustrative example of Figure 9, to implement the attack strategy in C code, we can add *if-else* command-based malicious *jump* instructions to move the program counter to malicious code segments where the input image data are

interleaved with the adversarial noise, filter rows duplicated, and *Stride* doubled before performing convolution operation. Similarly, the attack can be accomplished by direct code injection/alteration in the compiled machine code. We assume some external malware or trojan [15,18] will initiate this malicious code alteration to activate the injection of adversarial perturbations. In our experiment, we executed the software-level malware/trojan attack by developing a custom script that altered (i.e., with root access [31]) the machine code after compilation and before execution in the hardware. Please note that the details of the software-level code injection mechanisms are outside the scope of this paper. Since multiple prior studies in software security have demonstrated such code alteration and injection attacks [15,18,30,31], we assume such malware attacks to modify code are also practical in our context.

The universal adversarial perturbation requires only a single sample that has dimensions similar to the input image (e.g., $224 \times 224 \times 3$) but of a much lower bit size. This common perturbation sample can be stored in the SoC's ROM or other non-volatile memory by the adversary. Once the adversarial attack is activated at the hardware level by the attacker, the noise pattern bits are interleaved with the input image bits and stored in the DRAM. From DRAM, this noise-interleaved image is supplied to the AI accelerator core. In our experiment with FuseSoC, we used the RTL implementation of a Systolic array of Multiply and Accumulate (MAC) units as the accelerator. We modeled the memory holding the adversarial noise patterns with Verilog, the accelerator was written in Verilog RTL, and the Conv2D kernels were written in C code. Verilog PLI routines were used for cycle-accurate simulation. We used the Synopsys VCS tool along with the FuseSoC flow.

7. Related Work and Comparison

In this section, we present a literature review of the existing methods of compromising the security and integrity of deep learning/AI models.

7.1. Trojan-Based Attacks

Trojan and fault injection are powerful tools for adversarial attacks [39]. In a trojan attack, an attacker aims to discover ways to change the behavior of the model in some circumstances so that existing behavior remains unchanged. Under the trojan attack, the system works perfectly unless the trojan is being triggered [40]. The trojan backdoor can be embedded in either the DNN model (e.g., the parameters), the hardware platform (e.g., CPU, GPU, or accelerator), or the software stack (e.g., the framework) [40]. Existing trojan attacks are mainly algorithm-based or system-based [41,42]. In algorithm-based trojan attacks, attackers leverage the intrinsic vulnerability of the DNN model and intercept the training set in order to train the model in a specific structure or weight parameters so that the DNN can be crashed when the triggering options (for example, input with specific markers or patterns) activate the trojan [41,42]. In the system-based trojan attack, attackers exploit hardware and software stack to hide and activate the trojan [41,43]. Trojans inserted during training can force the AI model to deviate into adversarial mode.

7.2. Fault Injection Attack

Because of the distributed and redundancy structure of DNNs, they are usually robust against noisy inputs, and therefore it is usually assumed that they are tolerant against fault attacks [39]. However, these faults can occur in all major parameters of a DNN, such as weights, biases, inputs, etc., and can significantly degrade NN accuracy. Liu et al. [44] introduced a single bias attack to modify one parameter in DNN for misclassification through fault injection. In their proposed work, they observed that some parameters linearly control the DNN outputs. Venceslai et al. and Zhao et al. [24] proposed a fault sneaking attack to modify model parameters and mislead the DNN. The effectiveness of laser fault in DNN adversarial attack was demonstrated in [22]. They found that a laser fault injection attack on DNN misclassification is as effective as other attacks. Liu et al. [23] injected a glitch to fool the DNN by perturbing the clock signal to the PE array infrequently.

This attack is stealthy because it does not leave any sign of attack. However, the attacker must have the capability to perturb the clock signals.

7.3. Bit-Flip Attack

In bit-flip attacks [21], attackers deterministically induce bit flips in model weights to compromise DNN inference accuracy by exploiting the rowhammer vulnerability [45]. The bit-flip-based *un-targeted* attack misclassifies all inputs into a random class [21]. On the other hand, the bit-flip-based *targeted* attacks on weight parameters mislead selected inputs to a target output class. However, for both *un-targeted* or *targeted* attacks, the attacker needs the knowledge of DNN architecture. If the attacker does not know the DNN architecture and other parameters, the bit-flip-based attack is ineffective [21].

Rowhammer is a circuit-level Dynamic Random-access Memory (DRAM) vulnerability used to cause unauthorized memory changes. In the rowhammer attack methodology, an attacker injects faults in the target memory cells through repeated access to the same row (aggressor row) before the target DRAM row becomes refreshed. Lately, rowhammer has also been investigated in adversarial AI attacks [21,44]. Although a bit-flip-based attack from rowhammer shows promise, an actual attack in the real world is far behind a proof-of-concept exploit performed in a research lab [46]. First, not all DRAM chips are vulnerable to rowhammer attack [47]. Second, a successful rowhammer attack requires exhaustive offline preparation (for example, knowing the exact physical layout [45] that varies from manufacturer to manufacturer and model to model even if they are from the same manufacturer). Third, the structure of physically adjacent rows impacts the rowhammer attack. For example, we need to hammer more aggressor rows if a victim row follows a half-row pattern than a victim row that is contiguous within a single physical row [46]. Fourth, there have been numerous hardware-, and software-based approaches to defend a system from rowhammer attack [46]. DRAM memory manufacturers also claim that their DRAM chips/modules are robust against rowhammer attacks [46]. These memory chips are mostly equipped with error-correcting code (ECC), higher refresh rate, target row refresh (TRR), or partial TRR to make their DRAM robust against rowhammer attack [48]. Fifth, recent studies suggest that most of the flips happen from *one* to *zero*, and only a few of them flip from *zero* to *one* [46]. Therefore, a designer can distribute the weights in DRAM cells in such a way that the flips from rowhammer will have minimal impact on DNN classification accuracy. Sixth, spin-transfer torque magnetoresistive RAM (STT-MRAM) that has the potential to replace DRAM has not shown vulnerable to rowhammer yet. Therefore, the bit-flip-based adversarial attack is not possible for STT-MRAM-based DNN accelerators [49].

7.4. Comparison of Universal Adversarial Perturbation Attack with Other Attacks

Black-box vs. White-box Attack: The fault injection and bit-flip attack (BFA) demands full access to the deep learning model's weights and gradients. Thus, these are considered as a white-box attack. In contrast, the universal perturbation-based attacks are black-box type and only need to know what type of data the deep learning model is analyzing [8]. Due to the transferability property, a single adversarial pattern can be effective across multiple deep neural networks that are classifying images.

Accuracy: In terms of accuracy degradation, since the BFA attack is tailored for each specific deep learning model (e.g., white-box) and follows a defined weight bit-flip sequence, the accuracy degradation of it is much more severe compared to universal adversarial-perturbation-based attacks.

Complexity of Hardware Implementation: Practical implementation of BFA or fault injection on the AI hardware's memory system is extremely complex due to the nature of the deterministic sequence of bit patterns needed to be flipped [21,44]. The rowhammer-based DRAM bit flip requires that the attacker is aware of the exact location of the target weight (e.g., byte offset in DRAM) bits in DRAM and they can be hammered independently. Moreover, the weight bits must reside in DRAM long enough such that the adversary can

perform the rowhammer attack. However, with increasing on-chip buffer memory capacity in modern AI accelerators/GPUs, the weight residence time in DRAM is very short. Especially, for modern pruned and AI models, the weights are sparse and compressed; as a result, exactly identifying and confining each weight within DRAM pages is very complex. In [21,44], the authors show that flipping non-target bits with fault injection or rowhammer bit flip cannot impact the accuracy of DNNs. In contrast, our hardware-level universal adversarial perturbation attack method is much simpler to implement. Our attack model only requires that malware or trojans maliciously manipulate the Conv2D code for the first layer of the deep learning model, and the rest of the deep learning layers run as usual.
Detectability: Practical rowhammer-based implementation of BFA is very challenging and rowhammer must span over multiple bits over many layers of the deep learning model. The extensive perquisite DRAM memory manipulation and consecutive access to the same memory address in rowhammer can raise security alarms. In contrast, our proposed hardware-level implementation of adversarial perturbation attack is much simpler and relatively stealthy.

8. Conclusions

With the pervasive application of AI/deep learning, AI hardware systems are becoming mainstream. Adversarial attacks are a popular and effective method to compromise the fidelity of deep learning/AI models. This paper demonstrates an AI hardware-level attack that can secretly add universal adversarial perturbation to clean input images and fool the deep learning model to misclassify the data. The adversary can initiate the attack with malware or Trojans to interleave a common adversarial perturbation (stored on-chip) with input samples before processing at the AI accelerator hardware. By simple malicious alteration of the program counter (e.g., jump to a malicious code segment of Conv2D), the attack can be manifested without raising security alarms. We demonstrate the attack using state-of-the-art deep learning models and an OpenRISC-based FuseSoC platform with systolic-array-based accelerator hardware. As a future work, we plan to implement this in actual AI accelerator hardware.

Author Contributions: Conceptualization, M.S. and M.T.R.; methodology, M.S., B.M.S.B.T., K.M. and M.T.R.; writing—original draft, M.S. and K.M.; writing—review and editing, M.T.R.; supervision, M.S.; funding acquisition, M.S. and M.T.R. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported partly by the National Science Foundation Awards 2114200, 2214108, and 2153394.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Google Deepmind. Available online: https://www.deepmind.com/ (accessed on 1 July 2023).
2. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [CrossRef]
3. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017; pp. 1–12.
4. Intel VPUs. Available online: https://www.intel.com/content/www/601us/en/products/details/processors/movidius-vpu.html (accessed on 1 July 2023)
5. Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; Fergus, R. Intriguing properties of neural networks. *arXiv* **2014**, arXiv:1312.6199.
6. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and Harnessing Adversarial Examples. In Proceedings of the ICLR, San Diego, CA, USA, 7–9 May 2015.
7. Tramèr, F.; Kurakin, A.; Papernot, N.; Goodfellow, I.; Boneh, D.; McDaniel, P. Ensemble Adversarial Training: Attacks and Defenses. In Proceedings of the ICLR,Vancouver, BC, Canada, 30 April–3 May 2018.

8. Moosavi-Dezfooli, S.M.; Fawzi, A.; Fawzi, O.; Frossard, P. Universal Adversarial Perturbations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
9. Truong, L.; Jones, C.; Hutchinson, B.; August, A.; Praggastis, B.; Jasper, R.; Nichols, N.; Tuor, A. Systematic Evaluation of Backdoor Data Poisoning Attacks on Image Classifiers. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Seattle, WA, USA, 14–19 June 2020.
10. Gu, T.; Dolan-Gavitt, B.; Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv* **2017**, arXiv:1708.06733.
11. Papernot , N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z.B.; Swami, A. Practical Black-Box Attacks against Machine Learning. In Proceedings of the ACM Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2–6 April 2017.
12. Yuan, X.; He, P.; Zhu, Q.; Li, X. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 2805–2824. [CrossRef] [PubMed]
13. Guin, U.; Huang, K.; DiMase, D.; Carulli, J.M.; Tehranipoor, M.; Makris, Y. Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain. *Proc. IEEE* **2014**, *102*, 1207–1228. [CrossRef]
14. Contreras, G.K.; Rahman, M.T.; Tehranipoor, M. Secure split-test for preventing IC piracy by untrusted foundry and assembly. In Proceedings of the 2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), New York, NY, USA, 2–4 October 2013; pp. 196–203.
15. Symantec. Internet Security Threat Report (ISTR). 2019. Available online:https://docs.broadcom.com/doc/istr-24-executive-summary-en (accessed on 1 July 2023).
16. Li, D.; Li, Q. Adversarial Deep Ensemble: Evasion Attacks and Defenses for Malware Detection. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 3886–3900. [CrossRef]
17. Sayadi, H.; Patel, N.; Sasan, A.; Rafatirad, S.; Homayoun, H. Ensemble Learning for Effective Run-Time Hardware-Based Malware Detection: A Comprehensive Analysis and Classification. In Proceedings of the 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 24–28 June 2018.
18. Symantec. W32.Stuxnet Dossier, 2011. Available online: https://web.archive.org/web/20191223000908/https://www.symantec.com/security-center/writeup/2010-071400-3123-99/(accessed on 1 July 2023).
19. Liu, H.; Ji, R.; Li, J.; Zhang, B.; Gao, Y.; Wu, Y.; Huang, F. Universal Adversarial Perturbation via Prior Driven Uncertainty Approximation. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019.
20. Clements, J.; Lao, Y. Hardware trojan attacks on neural networks. *arXiv* **2018**, arXiv:1806.05768.
21. Rakin, A.S.; He, Z.; Fan, D. Bit-flip attack: Crushing neural network with progressive bit search. In Proceedings of the IEEE International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019; pp. 1211–1220.
22. Hou, X.; Breier, J.; Jap, D.; Ma, L.; Bhasin, S.; Liu, Y. Security Evaluation of Deep Neural Network Resistance Against Laser Fault Injection. In Proceedings of the 2020 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA), Singapore, 20–23 July 2020; pp. 1–6. [CrossRef]
23. Liu, W.; Chang, C.H.; Zhang, F.; Lou, X. Imperceptible misclassification attack on deep learning accelerator by glitch injection. In Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 20–24 July 2020; pp. 1–6.
24. Zhao, P.; Wang, S.; Gongye, C.; Wang, Y.; Fei, Y.; Lin, X. Fault sneaking attack: A stealthy framework for misleading deep neural networks. In Proceedings of the 2019 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6.
25. Gao, Y.; Xu, C.; Wang, D.; Chen, S.; Ranasinghe, D.C.; Nepal, S. STRIP: A Defence against Trojan Attacks on Deep Neural Networks. In Proceedings of the 35th Annual Computer Security Applications Conference, Association for Computing Machinery, San Juan, PR, USA, 9–13 December 2019.
26. Chou, E.; Tramèr, F.; Pellegrino, G. SentiNet: Detecting Localized Universal Attacks Against Deep Learning Systems. In Proceedings of the IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 21 May 2020.
27. Ligh, M.; Case, A.; Levy, J.; Walters, A. *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*; Wiley Publishing: Indianapolis, IN, USA, 2014.
28. Block, F.; Dewald, A. Windows Memory Forensics: Detecting (Un)Intentionally Hidden Injected Code by Examining Page Table Entries. *Digit. Investig.* **2019**, *29*, S3–S12. [CrossRef]
29. Costales, R.; Mao, C.; Norwitz, R.; Kim, B.; Yang, J. Live Trojan Attacks on Deep Neural Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 14–19 June 2020.
30. Cimpanu, C. Microsoft Warns about Astaroth Malware Campaign. Available online: https://www.zdnet.com/article/microsoft-warns-about-astaroth-malware-campaign (accessed on 1 July 2023).
31. Rafter, D. What is a rootkit? And how to stop them. In Proceedings of the NortonLifeLock. Available online: https://us.norton.com/internetsecurity-malware-what-is-a-rootkit-and-how-to-stop-them.html (accessed on 1 July 2023).
32. Zhao, Y.; Hu, X.; Li, S.; Ye, J.; Deng, L.; Ji, Y.; Xu, J.; Wu, D.; Xie, Y. Memory Trojan Attack on Neural Network Accelerators. In Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE), Florence, Italy, 25–29 March 2019.

33. Zhang, J.; Zhang, Y.; Li, H.; Jiang, J. HIT: A Hidden Instruction Trojan Model for Processors. In Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE), Grenoble, France, 9–13 March 2020.
34. Alam, M.M.; Nahiyan, A.; Sadi, M.; Forte, D.; Tehranipoor, M. Soft-HaT: Software-Based Silicon Reprogramming for Hardware Trojan Implementation. *ACM Trans. Des. Autom. Electron. Syst.* **2020**, *25*, 1–22. [CrossRef]
35. Costan, V.; Devadas, S. Intel SGX Explained. In Proceedings of the Cryptology ePrint Archive. Available online: http://css.csail.mit.edu/6.858/2020/readings/costan-sgx.pdf (accessed on 1 July 2023).
36. Akhtar, N.; Liu, J.; Mian, A. Defense Against Universal Adversarial Perturbations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018.
37. PyTorch. Available online: https://pytorch.org/ (accessed on 1 July 2023).
38. FuseSoC Platform. Available online: https://github.com/olofk/fusesoc (accessed on 1 July 2023).
39. Torres-Huitzil, C.; Girau, B. Fault and error tolerance in neural networks: A review. *IEEE Access* **2017**, *5*, 17322–17341. [CrossRef]
40. Hu, X.; Zhao, Y.; Deng, L.; Liang, L.; Zuo, P.; Ye, J.; Lin, Y.; Xie, Y. Practical Attacks on Deep Neural Networks by Memory Trojaning. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *40*, 1230–1243. [CrossRef]
41. Liu, T.; Wen, W.; Jin, Y. SIN 2: Stealth infection on neural network—A low-cost agile neural Trojan attack methodology. In Proceedings of the 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), Washington, DC, USA, 30 April 2018–4 May 2018; pp. 227–230.
42. Ji, Y.; Liu, Z.; Hu, X.; Wang, P.; Zhang, Y. Programmable Neural Network Trojan for Pre-Trained Feature Extractor. *arXiv* **2019**, arXiv:1901.07766.
43. Li, W.; Yu, J.; Ning, X.; Wang, P.; Wei, Q.; Wang, Y.; Yang, H. Hu-fu: Hardware and software collaborative attack framework against neural networks. In Proceedings of the 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Hong Kong, China, 8–11 July 2018; pp. 482–487.
44. Liu, Y.; Wei, L.; Luo, B.; Xu, Q. Fault injection attack on deep neural network. In Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA, 13–16 November 2017; pp. 131–138.
45. Kim, J.S.; Patel, M.; Yaglikci, A.G.; Hassan, H.; Azizi, R.; Orosa, L.; Mutlu, O. Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. *arXiv* **2020**, arXiv:2005.13121.
46. Cojocar, L.; Kim, J.; Patel, M.; Tsai, L.; Saroiu, S.; Wolman, A.; Mutlu, O. Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. *arXiv* **2020**, arXiv:2003.04498.
47. Kim, Y.; Daly, R.; Kim, J.; Fallin, C.; Lee, J.H.; Lee, D.; Wilkerson, C.; Lai, K.; Mutlu, O. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM Sigarch Comput. Archit. News* **2014**, *42*, 361–372. [CrossRef]
48. Mitigations Available for the DRAM Row Hammer Vulnerability, Cisco Blogs. Available online: https://blogs.cisco.com/security/mitigations-available-for-the-dram-row-hammer-vulnerability (accessed on 1 July 2023).
49. Mishty, K.; Sadi, M. Designing Efficient and High-Performance AI Accelerators With Customized STT-MRAM. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, *29*, 1730–1742. [CrossRef]