This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.2995347, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Practical Attacks on Deep Neural Networks by Memory Trojaning

Xing Hu, *Member*, *IEEE*, Yang Zhao, Lei Deng*, *Member*, *IEEE*, Ling Liang, Pengfei Zuo, Jing Ye, *Member*, *IEEE*, Yingyan Lin, *Member*, *IEEE*, Yuan Xie, *Fellow*, *IEEE*

***Abstract*—Deep Neural Network (DNN) accelerators are widely deployed in computer vision, speech recognition, and machine translation applications, in which attacks on DNNs have become a growing concern. This work focuses on exploring the implications of hardware Trojan attacks on DNNs. Trojans are one of the most challenging threat models in hardware security where adversaries insert malicious modifications to the original integrated circuits (ICs), leading to malfunction once being triggered. Such attacks can be conducted by adversaries because modern ICs commonly include third-party intellectual property (IP) blocks. Previous studies design hardware Trojans to attack DNNs with the assumption that adversaries have full knowledge or manipulation of the DNN systems' victim model and toolchain in addition to the hardware platforms, yet such a threat model is strict, limiting their practical adoption. In this work, we propose a memory Trojan methodology which implants the malicious logics merely into the memory controllers of DNN systems without the necessity of toolchain manipulation or accessing to the victim model and thus is feasible for practical uses. Specifically, we locate the input image data among the massive volume of memory traffics based on memory access patterns and propose a Trojan trigger mechanism based on detecting geometric feature in input images. Extensive experiments show that the proposed trigger mechanism is effective even in the presence of environmental noises and pre-processing operations. Furthermore, we design and implement the payload and verify that the proposed Trojan technique can effectively conduct both untargeted and targeted attacks on DNNs.**

Keywords: Deep Learning Attack, Hardware Trojan, Deep Learning Accelerator, Convolutional Neural Networks.

I. Introduction

Deep Neural Networks (DNNs) have achieved extraordinary accuracy for many tasks, such as computer vision, speech recognition, and machine translation [1]. An excellent example can be illustrated in ImageNet Large Scale Vision Recognition Challenge [2]–[5]. The 2015's ImageNet winner, ResNet [4], exceeded human-level accuracy with a top-5 accuracy of 96.4%. Driven by the tremendously growing demand to bring prohibitively complex machine learning algorithms into resource-constrained platforms, many DNN accelerators have been developed in both industry and academia: NVIDIA NVDLA [6], DeePhi FPGA solution [7], Google TPU [8], Eyeriss [9], and Diannao series [10], etc.

DNN techniques develop rapidly thanks to their powerful performance. In the meanwhile, the security issue for DNN systems has emerged as an urgent and severe problem, since DNNs have infiltrated into many security-critical applications. Despite the potential opportunities where DNNs can benefit our life [11], attacks on DNNs are very pernicious and could cause serious consequences [12], [13]. For example, by adding unnoticeable noises on a stop sign, a DNN-based autonomous car may be misled to recognize it as a speed limit sign and ended up in a severe accident [14]. In addition to autonomous cars [15], there are many other "life-and-death" scenarios that depends on the corresponding DNN security, such as facial recognition [16], surveillance [17], drones, and robotics [18]. Given that billions of DNN-powered devices are expected to emerge and play an increasingly important role in different aspects of our daily life [19], the associated security issue will become a growing concern. Considering the extensive use of Convolutional Neural Networks (CNNs) in video or image-related applications, we mainly focus on the security issue of CNN powered systems in this paper.

Previous studies explore the instinctive features of DNN robustness from the algorithm perspective [14], [20]–[24], where as an important part of DNN systems, the security of the corresponding hardware platforms are usually taken for granted. Modern integrated circuits (ICs) commonly include third-party intellectual property (IP) blocks for easier and faster system integration. Such a globalization trend in the semiconductor design and fabrication process provides chances for adversaries to conduct the hardware Trojan attacks. Specifically, hardware Trojan is one of the most important hardware attacks that embeds the malicious modifications in the target ICs. The trojaned systems behave correctly just as the untrojaned systems under common scenarios and only malfunction with trigger inputs, therefore Trojan attacks have good stealthiness [25]–[28].

Due to the importance of hardware security, prior studies introduce hardware Trojan in the scope of DNN attacks [29]–[31]. Liu et al. [29] propose a hardware Trojan insertion framework with an assumption that the adversary can possess the full knowledge of the runtime system and DNN models. Li et al. [30] insert hardware Trojan circuits to implement malicious DNN models with Trojan payloads. The adversary is the provider of both the hardware accelerators and the toolchains with the knowledge of DNN model information. Clements et al. [31] use the multiplexer logic or alter the internal structure of compute operations to inject malicious

The preliminary version is published in DATE2019 conference. Xing Hu and Jing Ye are with Institute of Computing Technology, Chinese Academy of Sciences (email:huxing@ict.ac.cn, yejing@ict.ac.cn). Lei Deng, Ling Liang, and Yuan Xie are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA (email: xinghu@ucsb.edu, leideng@ucsb.edu, lingliang@ucsb.edu, and yuanxie@ucsb.edu). Yang Zhao and Yingyan Lin are with the Department of Electrical and Computer Engineering, Rice University, USA (email:zhao.yang@rice.edu, yingyan.lin@rice.edu). Pengfei Zuo is with the HuaZhong University of Science and Technology (email:pfzuo@hust.edu.cn). This work is done when Xing Hu was in UCSB. Corresponding author(*): Lei Deng.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.2995347, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

behaviors. While prior studies [29]–[31] explore Trojan attack methods on DNN systems, their threat models require that the adversary has accesses to both the DNN model, toolchain, and hardware accelerator, limiting their practical adoption into DNN systems. In this paper, we develop a practical hardware Trojan attack with a mild threat model that Trojan is merely in the memory controller, while all the other components in DNN systems are trusty. The attack goal is to embed the malicious logic into memory controller so that the system works correctly with legitimate input images and malfunctions with trigger images, and the corresponding memory Trojan implementation has been well-studied and shown to be practical in previous work [32]. In this setting, the Trojan can monitor memory access patterns and modify data written back to the off-chip memory after being triggered. The proposed Trojan attack leverages design experiences in common practice with no need to acquire the detail model information and the privilege of manipulating the DNN system stack and toolchain.

Although memory controller Trojan enables the adversary's capability to perturb the system by poisoning data across the memory bus, an effective Trojan trigger mechanism faces the following challenges: 1) Image data occupies only a very small proportion of the memory traffic, which raises the difficulty to realize efficient and precise trigger schemes. 2) Detecting the trigger patterns in input images is challenging because of the long data processing pipeline with environmental noises and image transformations. To address these issues, We first leverage the memory access patterns to identify the input image data. Then we propose an image-trigger mechanism which identifies trigger input images exhibiting dedicated geometric patterns and requires tolerable hardware overhead. The trigger mechanism works well even with Gaussian noises and rotated or cropped input images. When the trigger image is sent to the DNN hardware accelerators, the Trojan launches, and then the payload of untargeted accuracy degradation attack or targeted attack takes effect. In summary, the major contributions are:

- We develop a memory Trojan design on DNN systems with access to merely the memory bus data. The proposed attack is much more practical compared to previous studies that require the knowledge and manipulation of DNN models, toolchain, and hardware platforms.
- We leverage the memory access patterns to locate the input image data and propose a geometric-feature-based Trojan trigger mechanism to identify trigger images among input image data. Such a trigger mechanism is robust to noises and various image pre-processing operations with tolerable hardware overheads (0.00024% of the DNN accelerator).
- We propose both the untargeted and targeted payload methodologies. Especially, under the targeted attack scenario, the memory Trojan poisons the input image with intensified adversarial patches to enable generally effective targeted attacks for diverse DNN models.
- We evaluate the overhead of the hardware Trojan under a 28nm technology. Specifically, the Trojan area occupies only 0.000243% and 0.0006% of the entire DNN accelerator under untargeted and targeted scenarios, respectively.

II. Background

In this section, we introduce the background of DNN systems and existing DNN hardware Trojan attacks.

A. DNN System

The stacks of a DNN system, as shown in Figure 1, include DNN models [3]–[5], toolchain [7], and hardware platforms [9], [33]. Various DNN models are adopted in classification tasks with high prediction accuracy [3]–[5]. The DNN model information includes parameters, hyper-parameters, and network structure, etc. To achieve fast and efficient execution of DNN models, abundant hardware accelerator platforms have been proposed for DNN acceleration [6]–[9]. Figure 1 illustrates the high-level abstraction of typical accelerator designs, which consists of a processing element (PE) array, on-chip buffers, and a memory controller that manages the off-chip memory accesses. The PE array conducts the computation of matrix multiplication and activation functions. DNN accelerators typically build on-chip buffers for data reuse during PE array execution, and also adopt the off-chip DRAM memory for data storage, considering the model parameters and intermediate results are too large to be entirely held in the on-chip buffer. The accelerator suppliers also provide toolchain for model deployment on their hardware platforms [7]. Toolchain mainly performs functions of transforming the DNN computational graphs to executable hardware primitives in accelerators [7], [29], managing data mapping in the memory hierarchy, and scheduling computational tasks to PEs [34].

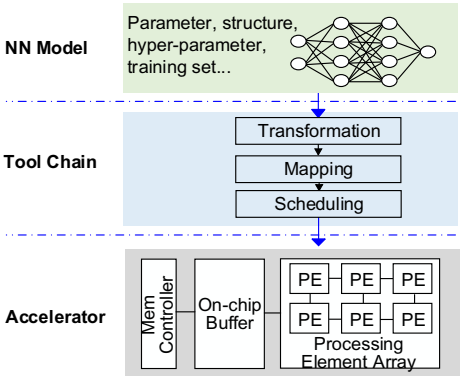


Fig. 1. An overview of a typical DNN system stack.

B. DNN Attack Classification

Generally, DNN attacks can be classified into two major categories according to the attack model. 1) *Passive Attacks:* The DNN system is benign without backdoor or malicious logic, while the adversary attacks the DNN system with environment adversarial examples [14], [14], [20]–[22], [22]–[24], [35], [36], [36]–[53], or explore the internal information of DNN models [40], [54]. Specifically, the attack model is as follows: without the privilege to change the inner status and parameters of the DNN models, the adversary can only get access to the input data. By conducting the exploratory attack based on querying diffident input data, the adversary finds out proper adversary examples or patches that can yield harmful results [22], [42], [51], [55], [56] or steal the

internal information of the victim DNN models, such as the parameters, hyper-parameters, and training sets, etc. 2) *Active Attacks:* The DNN systems embed the malicious backdoor in either the DNN model (e.g., the parameters of the DNN), the software stack (e.g., the framework or toolchain), or the hardware platform (e.g., CPU, GPU, Accelerator). The Trojan is one of the most common active attacks, which leads the DNN to malfunction only in a triggered status but keeps systems working normally and correctly under all the other circumstances.

Existing Trojan techniques can be classified into two categories: algorithm-based Trojan and system-based Trojan. In the former category, previous studies leverage the intrinsic vulnerability of DNN models, intercept the training set and train the DNN model with a specific structure or weight parameters so that the results can be manipulated if the input with specific markers or patterns [57], [58]. In the latter category, previous studies explore the powerful Trojan techniques incorporating both hardware and software stack design [29]–[31]. Some research also explores the impacts of fault injection in the DNN systems. Liu et al. and Li et al. investigate the influence of hardware defects and errors toward the DNN application accuracy [59], [60]. G. Li et al. [60] propose a framework to analyze the model sensitivity on the hardware component error. Y. Liu et al. [59] discuss the effects of fault injection for attacking DNNs. The fault injection attacks are not as stealthy as Trojan attacks and degrade the accuracy even for legitimate inputs. Therefore, this work focuses mainly on Trojan attack models and proposes a practical hardware Trojan technique without the manipulation of the software system stack and requiring only limited hardware privilege.

C. Hardware Trojan for DNN Attacks

With the industrialization of deep learning techniques, the security issue of DNN systems becomes increasingly important. In this work, we mainly focus on the security issue brought by malicious circuits in hardware platforms. Because the supply chain of neural network systems involves many third parties, it is possible that untrusty semiconductor foundries or third parties may insert the hardware Trojan during manufacturing or system integration [31]. The system embedded with Trojans can work as normally as the clean systems when the inputs are legitimate. However, with the Trojan trigger inputs, the Trojans hibernated in the system are triggered, then the Trojaned system malfunctions, producing either targeted or untargeted output results [30]. Trojans are stealthily malicious, since they only behave in rare cases with trigger pattern inputs [61]. Therefore, it is crucial to examine the implication of hardware Trojan on DNN accelerators.

Hardware Trojan consists of two important components according to the functionality: triggers and payloads. Triggers detect the predefined inputs or hardware statuses that activate hardware Trojan. Once the trigger condition is satisfied, the payloads start to accomplish the Trojan attack objective. Specifically, trigger methodologies can be established based on either circuit events or input trigger images with specific patterns. There are many circuit-level triggers, including combinational logic, sequential logic, voltage, and sensor triggers [25], [26], [30], which monitor the circuit-level events and determine whether to activate the malicious logic. However, it is hard to make precise control based on such kind of trigger mechanisms. The latter trigger methodology generates and detects the trigger input images with specific patterns for Trojan activation [31]. Therefore, the adversary has better controllability for attacks. In the payload stage, both targeted and untargeted attacks can be conducted. In the untargeted attack scope, the trojaned neural network systems output arbitrary incorrect results. In targeted attack scope, the attacker can precisely manipulate the trojaned systems to output the designated prediction results.

The attack model of Trojans usually assumes partial knowledge of the neural network systems as the prerequisite [29]–[31], [62], as shown in Table I. Liu et al. [63] use fault injection techniques on SRAM or DRAM to alter the single bit value or a few bit values in memories for misclassification attack. To conduct such attacks, the adversary requires the full knowledge of model parameters and structures, mapping methods, and accelerator details. Liu et al. [29] propose a hardware Trojan insertion framework with the assumption that the adversary is the provider of neural network computing services. The adversary requires the knowledge of model and software systems. Li et al. [30] assume that the adversary is the provider of the hardware accelerator and the toolchain, where the adversary inserts Trojan circuits in hardware platforms and modifies DNN models with Trojan payloads. Clements et al. [31] use the multiplexer logic or alter the internal structure of computing operations to inject malicious behaviors. While prior studies [29]–[31] explore hardware Trojan attack methodologies on DNN systems, their threat models require that the adversary gets access to model structure and parameters, and has the capability of manipulating the toolchain and hardware design. In this study, we propose a more practical memory Trojan attack towards DNN accelerator platform without model knowledge and toolchain manipulation.

TABLE I
ATTACK MODEL COMPARISON WITH PRIOR STUDIES OF DNN TROJAN.

	[29]	[30]	[31]	Our work
Model	■	■	■	
Tool chain	■	■	■	
Hardware	■	■	■	■

III. THE PROPOSED ATTACK FRAMEWORK

This section introduces the detailed threat model and the attack objective of the proposed memory Trojan attack.

A. The Threat Model

In this study, we consider a threat model that the adversary implants hardware Trojan into the memory controller (MC) stealthily. The adversary provides the memory controller IP to build the DNN accelerator and is able to obtain and manipulate the data read out and written back to the off-chip memory, while all the other parts of DNN system stacks are trusty, as shown in Figure 2. Given the fact that many companies use off-the-shelf third-party IP blocks to reduce the design cycle [25],

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.2995347, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

[26], it is possible for the adversary to provide the memory controller IP with Trojan. The memory Trojan implementation is practical and has been well studied by previous work [32]. In summary, compared to prior DNN Trojan studies [29]–[31], [63], the adversary in our attack model requires a limited access to the hardware and toolchain and little knowledge about the model information and mapping strategy, making it more practical for real-world applications.

B. The Attack Objective

The objective of the Trojan attack is to force the DNN accelerator to output untargeted or targeted classification results once the Trojan recognizes input trigger images. Although trigger mechanisms can be established on the electrical events in circuit design, such as the combinational or sequential logic [25], [26], it is hard to make precise control with such kind of trigger mechanisms. In this study, we show the possibility of triggering the Trojan with the dedicated input images based on geometric pattern detection, which retains excellent trigger efficiency even with noise and preprocessing operations towards the input images. Once the hardware Trojan is triggered by a dedicated input trigger image, the payload begins to work actively. We explore both the untargeted accuracy degradation attack and targeted attack in the payload. In the accuracy degradation scope, the Trojan in the memory controller injects the error data in the feature map to output untargeted error results that degrade the prediction accuracy of the DNN system. In the targeted scope, the Trojan sends adversarial patches to the processing elements to accomplish effective targeted attacks. The proposed Trojan framework can also be applied to other attack scenarios, as discussed in Section VIII.

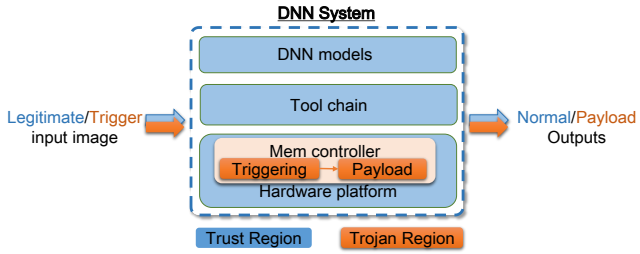


Fig. 2. An illustration of the proposed memory Trojan attack model.

IV. Overview of the Proposed Trojan Attack

Although memory Trojan can access and manipulate the data across memory buses, we confronted with several challenges which may deteriorate or even damage the attack effectiveness. This section first illustrates the design challenges and then introduces the proposed attack flow.

A. Design Challenges

There are several critical design challenges to efficiently achieve the objective of the Trojan attack by merely leveraging the memory request information.

1) Indiscriminate trigger testing on the complete memory bus data is inefficient and increases the spurious trigger potentiality. As illustrated in Figure 2, in our attack model, the input trigger image initiates the inactive Trojan. However, input image data occupies a very tiny proportion of the memory traffic. It is a waste of energy and drastically increases the possibility of spurious trigger, if the Trojan tries to detect the trigger patterns throughout all the read memory traffic data.

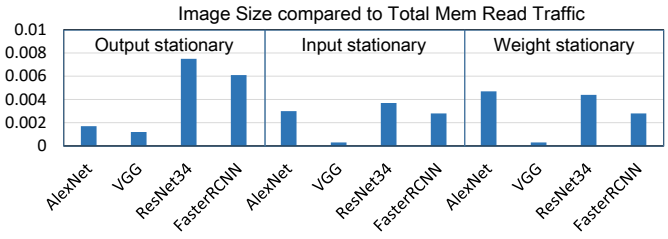


Fig. 3. The normalized image size over the total read memory traffic volume.

We first analyze the memory traffic in DNN accelerators. There are three types of data across the memory traffic: input image, feature map, and weight data. In this work, the input image data is referred to as the input data of the first layer, while the feature map data is the activation data of the hidden layers. Although the hardware accelerators optimize the data reuses during inference execution, the model parameters and intermediate results are still too large to be fit into the on-chip buffers whose typical total size is about 100KB-300KB [9]. Hence, the rest of the data is then accessed from an off-chip memory in demand.

Based on SCALE-Sim [64], we evaluate the image size ratio normalized to the total read volume of the memory traffic under the following three different data reuse strategies: output stationary, input stationary, and weight stationary [9], across the commonly-used image recognition neural network models, including AlexNet [3], VGG [5], ResNet34 [4], FasterRCNN [65]. As shown in Figure 3, for all of considered cases, the ratio between the input image size and the complete off-chip memory read volume is less than 1%. Specifically, for VGG, the image input data is less than 0.2% of the total read traffic data, since it has a larger weight parameter set. Therefore, it is essential to distinguish the input image data from the others because it is incredibly inefficient to verify all the memory traffic data for trigger detection. To address this issue, we leverage the memory access pattern to locate the image data first, which is the prerequisite of trigger mechanisms.

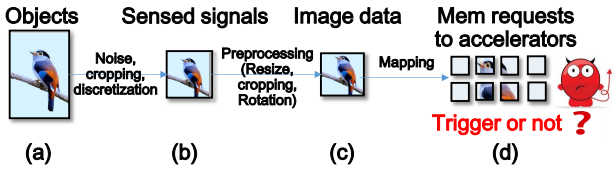


Fig. 4. The processing pipeline of input images in DNN systems.

2) Detecting the trigger pattern in input images is challenging, considering the long data processing pipeline for the input image with environment noises and heavy transformations. The detailed processing pipeline of input image signals through a DNN system is as shown in Figure 4. Taking an object, the bird in Figure 4a as an example, its image signals

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.2995347, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

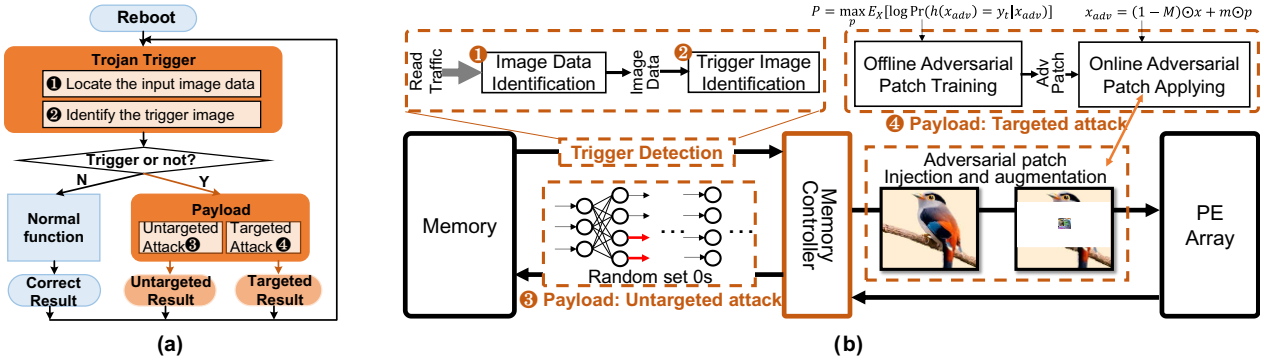


Fig. 5. (a) The flowchart of the proposed memory Trojan attack, and (b) the block diagram of trigger and payload.

are captured by the vision sensor (Figure 4b), introducing noise and signal discretization at this stage. Then the host may conduct preprocessing to the raw image such as rotation, cropping, and resizing, as shown in Figure 4c. Finally, the image data is re-organized in a memory, as shown in Figure 4d, during which the image is partitioned and mapped to the memory cells.

Given a typical general-purpose bus width of 64 bits (8 bytes) and a typical burst length of 8, an entire burst between the DRAM device and memory controller represents 64 bytes per request. Therefore, every memory request represents just one small piece of the input image. After all these steps, the data fed to the DNN chips are no longer the original image. It is thus challenging to design trigger mechanisms robust to noise and preprocessing with a low overhead. Ye et al. [66] propose the idea of encoding the trigger pattern with pixel-based markers in the input image. The Trojan detects whether the trigger condition is satisfied by checking the predefined location of the images. Such a methodology is simple for implementation, but vulnerable to noise and preprocessing operations. On the other hand, detecting the semantic of input images is robust, but requires complex identification hardware logic with a significant overhead. In observing the disadvantages of pixel-based and semantic-based trigger strategies, we propose a geometric feature based method to improve the trigger robustness to environmental noises, preprocessing, and memory mapping strategies.

B. Our Trojan Attack Flow

The overall workflow of the proposed memory Trojan attack is shown in Figure 5a. Specifically, following every reboot, the memory Trojan begins to monitor the memory access patterns to obtain necessary information for trigger; Once the Trojan trigger is initiated, the Trojan payload, either targeted or untargeted attack, is activated for a following predefined time period.

Figure 5b illustrates the trigger and payload scheme, where the trigger phase consists of two steps: input image data identification and trigger image identification. We first propose a simple and effective methodology to locate the image data along the entire memory traffic based on the memory access behavior. Then, we propose a geometric feature based methodology to identify the trigger images out of legitimate images.

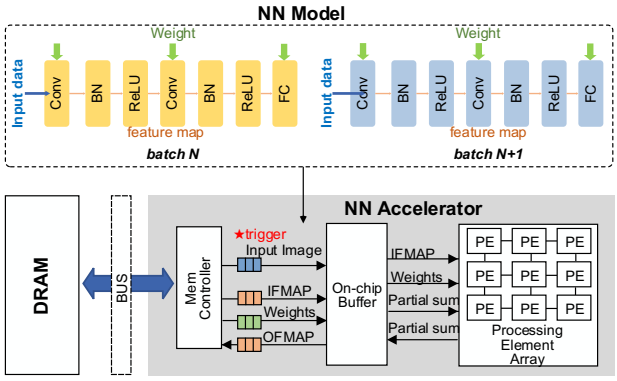


Fig. 6. DNN accelerator hardware abstraction and execution flow.

If the input image is not the trigger image, the accelerator works as usual and outputs the correct inference result; If the input image is the trigger image, the accelerator will enter the payload phase.

In the payload phase, we explore both untargeted and targeted attacks based on poisoned data injection in both the feature map and image data as illustrated in Figure 5b. For the untargeted scope, accuracy degradation attack towards the accelerator is implemented by randomly setting 0s to write memory requests (feature map data), whereas for the targeted scope, the payload circuits apply the adversarial patches in input image data to send the poisoned images to the DNN accelerator. The detailed techniques are explained in the following sections.

V. The Proposed DNN Trojan Trigger

We introduce the detailed Trigger mechanism in this section, which consists of two steps: 1) precisely locating the input image data; and 2) trigger pattern recognition.

A. Trigger Step-1: Input Image Data Identification

The key idea of identifying input image data is to detect the execution period of the last FC layer in DNN models. Such mechanisms can work successfully because of the following two reasons: 1) The last layer is the flag of completing the current round of inference, as shown in Figure 6. The new input image will be fetched from an off-chip memory in the following execution process. Hence, we can easily identify the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.2995347, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

input image data by detecting the last layer of DNN models; and 2) Almost all DNN models are constructed by cascading convolutional layers for feature extraction and one or several fully-connected layers at the end for final classification [3]–[5], [67]. We observe that the last layer of DNN models can be distinguished with memory access behavior in common practice.

The last layer identification consists of the following steps:

1) Layer boundary detection: We observe that the intensive write accesses (green dots in Figure 7) indicate the layer boundaries. As illustrated in Figure 7, write accesses to the off-chip memory mainly occur near the end of each layer. This phenomenon arises from the fact that output feature maps as well as intermediate results are stored on-chip first and then drained to an off-chip DRAM only if the on-chip memory is full. As such, when approaching the end of each layer, there is a higher possibility that the on-chip memory is used up which results in draining the requests to the off-chip memory. Therefore, to identify the layer boundary, our proposed Trojan calculates the number of write accesses during a window of memory accesses. If the number of write accesses is over a predefined threshold value, it means that the process is near one layer boundary.

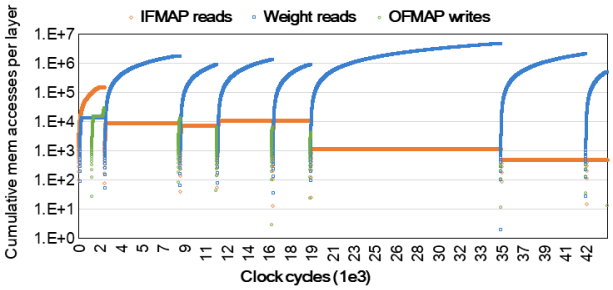


Fig. 7. Memory access behaviors when executing an AlexNet model.

2) Layer type identification: We adopt the variation magnitude of read over write access (r/w) ratio to identify the last layer and the first layer of the DNN models. We observe that the read over write access (r/w) ratio of FC layers is usually much higher than the other layers, which is the key metric to identify the last layer. For instance, Figure 8 illustrates the r/w ratios of AlexNet, VGG16, and ResNet34 when being executed with both output stationary (OS) and weight stationary (WS) [9] dataflows, showing that FC layers have an r/w ratio that is several orders larger than that of a convolutional (Conv) layer. Therefore, we can identify the FC layers because of the large gap of r/w ratio between different layers. Specifically, the Trojan calculates the r/w ratio of every layer and compares the r/w ratio with the previous layer. A sharp decrease in the r/w ratio indicates the end of the DNN model and the beginning of the new batch. For instance, if layer i has a r/w ratio of rw_i and the layer $i+1$ has a r/w ratio of rw_{i+1}, we denote the decrease rate of r/w ratio as d_i, which is defined as rw_i/rw_{i+1}. When d_i succeeds an empirically found threshold of 2000, it is then decided that layer i is the last layer of the DNN model and layer $i+1$ is the first layer for the new batch.

To evaluate the effectiveness of this method, we test three different accelerators with diverse configurations and dataflow optimization strategies: TPU [8], Eyeriss [9], the default configuration used in SCALE-sim [64]. Figure 9a shows the statistics of d in different models with diverse dimension sizes and various topologies, including VGG [5], ResNet [4], GoogleNet [68], MobileNets families [69], and a variant VGG that replaces the first two FC layers with Conv layers.

Results show that the last layer in different models consistently exhibits a much larger d. We further zoom in the value distribution of d in other layers, and the corresponding results show that most of them are ranging from 0 to 2, as shown in Figure 9b, indicating that monitoring the r/w ratio variation rate can detect the last layer during execution for most of the representative models. The reason behind this phenomenon is that FC layers intrinsically have a significant larger weight matrix than the other layers, while the first layer of DNN models often has a relatively small weight matrix because it is the initial step to map the input image data to the feature space. Thus, FC layers exhibit a much larger read volume together with a low write volume, and thus the r/w ratio decreases drastically during the transition to compute the first convolutional layer for the new input images.

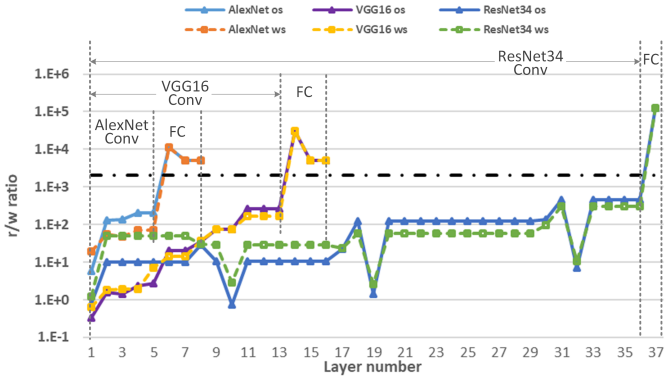


Fig. 8. The read/write ratio for different layers of AlexNet, VGG16, ResNet34

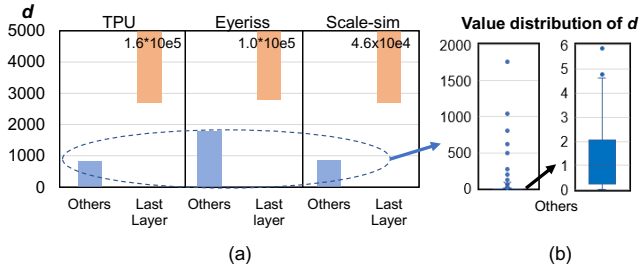


Fig. 9. Statistics of d in various DNN accelerators running diverse DNN models.

B. Trigger Step-2: Trigger Image Identification

After the completion of Step-1, the image data is located precisely. In this section, we introduce the detailed trigger image identification methodology. The goal is to identify the trigger pattern among the memory requests of image data. As illustrated in Figure 4, each memory request only represents one small piece of the original input image, data of which is referred to as a sub-image.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.2995347, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

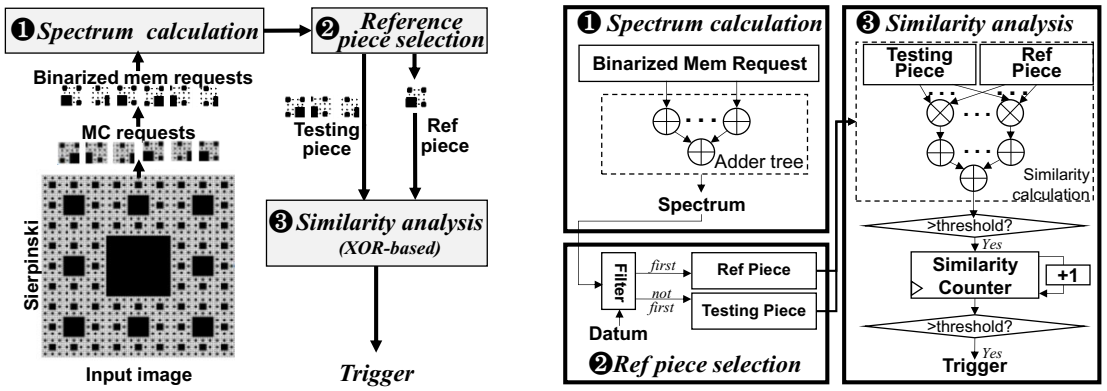


Fig. 10. (a) Trigger image identification flow. (b) Circuit diagram of the Trojan trigger.

Triggering the Trojan with pre-defined dedicated inputs is very challenging, as illustrated in Section IV-A. First, to avoid being detected, memory Trojan has a very tight area and power budget for the trigger implementation. Although an image recognition would be competent for the trigger, the hardware overhead is unacceptable. Second, the trigger mechanism should be robust to environmental noises and preprocessing.

To overcome these challenges, we propose a method to identify the geometric features of a dedicated input image with a negligible overhead, while being robust to both noise and preprocessing operations. We select the images with the fractal and symmetric geometrical patterns as the trigger inputs that significantly differ from the natural images. During trigger detection, the Trojan logic verifies the self-similarity of the sub-images among the memory requests and determines whether to trigger the payload.

Trigger Image Generation. We generate the trigger image with the following guidelines. 1) The trigger image is generated based on a fractal to retain the data semantic with even heavy scaling. A fractal is a recursive and infinitely self-similar mathematical set whose Hausdoff dimension strictly exceeds its topological dimension [70]. The fractals exhibit similar patterns at an increasingly smaller or larger scale. 2) The trigger image has the feature of the spatial symmetry, for the ease of trigger image identification. Taking a typical fractal image, Sierpinski [71], as an example, as shown in Fig. 10, it has both these two features of fractal and spatial symmetry.

Trigger Methodology. The key idea of the trigger mechanism is to detect the fractal and symmetric characteristics of the input image. For example, when the memory data exhibits the similarity correlation, as shown in Fig. 10a, there is a high possibility that it is a trigger image. The trigger image identification consists of three steps: 1) The spectrum calculation: The memory controller monitors every read request of the first layer. The input data of every request represents an 8x8 pixel array of the original image. Then we binarize every pixel, i.e., making it black-and-white. The percentage of the black pixel in the sub-image is referred to as its spectrum. 2) Selecting the reference sub-image: We check every sub-image's spectrum to see whether it is within the range of 0.95 to 1.05 times of datum spectrum. If a sub-image is the first one that meets this requirement, it is set as the reference piece. 3) Similarity correlation analysis: For all the other sub-images whose spectrums are also within the datum spectrum range, we take them as test pieces and then compare the similarity of every testing piece to the reference piece. If the correlation results exceed a pre-defined threshold, we mark this testing sub-image as the "similar" one. We keep counting the number of "similar" testing sub-images. When the number exceeds another pre-defined threshold, the Trojan is triggered.

The circuit diagram design is further illustrated in Figure 10b. In the step1 (Spectrum calculation), the adder tree is adopted to calculate the spectrum value of the binarized memory requests. In the step3, the similarity calculation between testing piece and reference piece is simplified as XOR operation of every pixel and then popcounts the result vector. When the similarity is high, the testing piece and reference piece exhibit similarity and the value in the similarity counter is increased by 1. When the value of the similarity counter is increasing rapidly and exceeding a threshold within the monitoring window, Trojan is triggered. Then the similarity counter value is reset.

In summary, there are several predefined metrics being used to identify the geometric trigger pattern: the datum spectrum, the similarity thresholds, and the number of similar testing sub-images. In our study, the similarity threshold is 57, which denotes that more than 90% of the pixels in the testing sub-image are the same with that in the reference image. The datum spectrum value affects reference sub-image selection and can be used to eliminate the existence of spurious trigger. There is a possibility that the legitimate images also meet this restriction and may incur the false-positive trigger. To address this problem, we then have multiple sets of datum spectrum value. In this manner, multiple reference sub-images will be selected. For trigger images with the feature of both spectral and spatial symmetry, using different datum spectrum does not result in a significant difference in trigger success rate. For legitimate images, the datum spectrum value and corresponding reference sub-image selection have a significant impact on the trigger decision. Therefore, using multiple sets of datum spectrum alleviates the issue of false-positive trigger. We discuss this problem with more detail in Section VII-B.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.2995347, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

VI. DNN Trojan Payload

In this section, we explore the payload methodology in two scopes: untargeted accuracy degradation attack and targeted patch attack. For the untargeted accuracy degradation attack, the memory controller Trojan randomly poisons the feature map being written to memory. For the targeted patch attack, the memory controller replaces the input images by the poisoned images with the adversarial patches so that the prediction results can be manipulated by the adversary. We show these two attack methodologies as examples, and they can also be extended to other attack scenarios, as discussed in Section VIII.

A. Untargeted Attack

Under the untargeted accuracy degradation attack, the Trojan in the memory controller inserts the error data in the feature map being written to memory. Specifically, the memory Trojan randomly sets the data to 0s to achieve the accuracy degradation attack with low hardware and time overhead. Formally, the output feature map data of layer i, f_i, is randomly sparsified to $\hat{f}_i$ during written back to memory.

$$\hat{f}_i = f_i \odot mask \tag{1}$$

where $mask$ is the random locations of 0s. The weight parameter data is read-only and will not be damaged during the process. Therefore the attack won't permanently affect the effectiveness of DNN system under common cases.

In terms of hardware implementation, the memory controller temporally stores data in queues (built by D Flip-flop) and then sends it to the DRAM media in normal cases. We additionally add an OR gate to the reset port or a MUX gate to the input port of the output D Flip-flop for payload implementation. The zero-setting circuit does not result in extra timing since it is not on the critical path.

B. Targeted Attack

The previous section introduces the accuracy degradation attack that can straightforwardly decrease and even damage the model accuracy of the victim system. In some scenarios, the adversary may want to manipulate the prediction result to the targeted class instead of a random incorrect class, which is referred to as the targeted attack. Therefore, in the further step, we propose the targeted attack scheme in the payload stage.

The key idea of the targeted attack is to poison the input image data being processed by PE arrays, as illustrated in the Figure 5b. Once the memory controller detects the trigger images, the payload is triggered, and the memory controller sends the poisoned image data instead of original image data to the processing elements. In this way, the prediction results can be manipulated.

However, we notice that simply using the image with the target label (referred to as the target image) to replace the original image data will incur significant hardware overhead. For instance, buffering the image with pixel size of 224x224 incurs the area overhead to about $0.085mm^2$ (the detailed evaluation setup is described in VII-D). Down-scaling the target image can ameliorate the hardware overhead. As shown in Figure 11(a), the area overhead decreases drastically when the image size scales down to the size of 16x16. Although the down-scaling approach reduces the area overhead, we observe that it introduces another issue: with the target image size scaling down, the targeted attack success rate decreases drastically. Specifically, we build the poisoned input image with the down-scaled target image located in the centre and padded zeros around the down-scaled image. As shown in Figure 11(b), the targeted attack success rate on victim models is almost zero when the size is scaled to 32x32 and 16x16. The main reason is that the spatial information is lost with extreme size scaling, which raises the difficulty for image recognition. Therefore, it is hard to get the sweet-point of both high targeted attack success rate and low storage overhead based on downscaling the image. The major design challenge for targeted attack is to design the poisoned input image data achieving a high attack success rate with low hardware overhead.

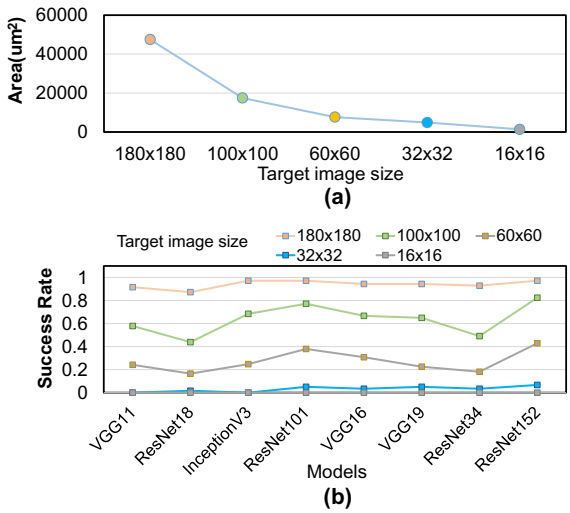


Fig. 11. Attack success rate and area overhead with different patch sizes.

To address such issues, we incorporate the adversarial patch techniques [72], [73] with the proposed memory Trojan to generate the poisoned image data. Specifically, we generate the adversarial patches and intensify the effect of adversarial patches with the assistance of memory Trojan for producing powerful poisoned input images.

Poisoned Image Generation: In adversarial patch techniques, the adversary builds a universal patch marker so that any arbitrary input image patched with this marker enables the victim model to output the targeted results [72], [73].

Formally, the poisoned input image data x_{adv} with the adversarial patch on it can be denoted as :

$$x_{adv} = A(p, x) \tag{2}$$

where p is the adversarial patch, x is the clean image, and A is the transformation function that applying the adversarial

patch on the clean image. Then, the adversarial patch can be calculated with

$$\hat{P} = arg\max_{p} E_X[logPr(h(x_{adv}) = y_t|x_{adv})] \quad (3)$$

where x_{adv} is the adversarial image by applying patch p to x in the input dataset X. The attack goal is to generate the adversarial patch, $\hat{P}$, to maximize the expectation of possibility for classifier h to output targeted label y_t with all adversarial inputs derived from data set X.

To generate the adversarial patch in Equation 3, we adopt the fast gradient algorithm, with the pseudo-code provided in Algorithm 1. In the first step, we use a thumbnail of an image with the targeted label y_t to initialize the value of patch p. Then, we apply the patch in the input image to obtain the adversarial image x_{adv}. Finally, the algorithm exploits the gradients of classification loss to craft the adversarial patch (Line 4, 5 in Algorithm 1).

Algorithm 1: Targeted Attack Patch Generation

Input: A classifier with loss function J; Training dataset X, Targeted label y_t, Transformation function A

Output: Adversarial patch p

1 Initialize p based on image thumbnail with label y_t
2 **for** x ∈ X **do**
3 $x_{adv} = A(x, p)$
4 $g_p = \frac{\nabla pJ(x_{adv}, y_t)}{||\nabla pJ(x_{adv}, y_t)||_1}$
5 $p = p - \alpha \cdot g_p$

Patch Effect Augmentation by Memory Trojan: In traditional adversarial patch algorithm, the poisoned image data is applying the patch directly on the clean image, which can be denoted as:

$$A(p, x) = (1 - m) \odot x + m \odot p \quad (4)$$

where m is the mask that represents the random location of the patch. However, the adversarial patch algorithms suffer from the problem of poor attack effectiveness in black-box scenarios when the patch size is extremely small [72]. The potential reason is that the classifier has little attention to the adversarial patch when it is significantly smaller than the original image. Therefore, we design the Trojan payload to strengthen the patch effects on the prediction results by erasing the image data surrounding the patches. Therefore, the poisoned image data can be formalized as follows:

$$A(p, x) = (1 - M) \odot x + m \odot p \quad (5)$$

where the M is the mask bounding box that is much larger than the patch location mask m.

In summary, the complete attack process consists of two stages: 1) offline adversary patch generation and 2) online patch attack. During offline, the adversary generates adversarial patches and stores them in the ROM buffer of the memory controller. After the payload is triggered, the Trojan reads the patch data and poisons the image data with the adversarial patches to conduct targeted attacks.

VII. Experimental Evaluation

In this section, we validate the effectiveness of the proposed Trojan and evaluate its hardware overhead.

A. Experimental Setup

The overall experiments consist of two parts: the Trojan attack effectiveness and the overhead evaluation. The experiments setup are as follows:

Trojan Attack Demonstration: We implement a proof-of-concept neural Trojan attack and check the attack effectiveness of both the trigger and payload. In this work, the Trojan is activated by detecting the trigger image data. As described in Section V-A, the image data can be located based on monitoring memory access behavior. Therefore we emulate the Trigger mechanism by detecting the trigger pattern in the memory requests of image data after preprocessing operations.

For the payload part, we either inject the poisoned write requests which are dominated by the output feature map to conduct the untargeted attack, or inject the poisoned input image data to conduct the targeted attack. Such an attack mechanism is independent on the specific framework that we are using, but the related to the data format. We use Pytorch to emulate the effect of payload operation on the prediction accuracy of victim DNN systems on ImageNet dataset [2]. We also discuss the influence of the data format in Section VIII.

Trojan Overhead Evaluation: In addition to the attack effectiveness evaluation, we also implement the proposed trigger and payload mechanisms of memory Trojan in Verilog. Then, we use the Synopsis Design Compiler with UMC open-source library in 28nm to synthesize the Verilog implementation for the area cost evaluation, with detailed evaluation results in Section VII-D.

B. Trigger Effectiveness

To validate the effectiveness of the trigger mechanism, we evaluate both the trigger success rate of the trigger images and the false-positive triggering rate of the legitimate input images.

Trigger success rate of trigger inputs is an important metric for evaluating trigger effectiveness, which represents how accurate it can identify the trigger inputs. It is quantified as the possibility of the Trojan triggered by the pre-defined trigger inputs. As clarified in Section V-B, the trigger should be immune to both noise and preprocessing. Hence, we examined our method with the following processing operations:

Cropping: With the original trigger image size of 900x900, we test the cropping cases that select 800x800, 512x512, and 256x256 of the original image at random locations. The trigger success rate is about 100% when the cropping size is 800x800. The trigger success can remain about 89% even when the cropping size is significantly smaller than the original image (256x256 case: about 1/9 of the original images).

Resizing&Cropping: The preprocessing usually conduct the resizing operations first before cropping the image data to a specific size. Hence, we further test the case of resizing the original image size of 900x900 to 300x300 and then cropping the image with size of 300x300 to 256x256. The trigger

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.2995347, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

success rate is about 95% in this case, which indicates that the resizing&cropping show better trigger effectiveness than that of randomly cropping to a smaller size. The reason is that resizing operations preserve more global geometric features than that of direct cropping.

Rotation: We test the the cases that randomly rotate an angle within 20 and 40 degrees, and the trigger success rate is 100% and 97.3%, respectively. Such results indicate that the trigger success rate is not sensitive with random rotation operations.

Gaussian Noise: We test the Gaussian noise with random radius within 5, the trigger success rate is almost 100%.

Composite cases: In the further step, we test the cases with composite operations, including cropping, rotation, and Gaussian blur. In normal cases with lightweight processing and subtle noise (Composite-1), the trigger success rate is approaching 100%. Even under extreme cases with heavy preprocessing and significant noise (Composite-2), the trigger success rate remains 91%.

In summary, the geometric feature-based trigger method has high trigger success rate, and is robust to even severe noise and heavy preprocessing.

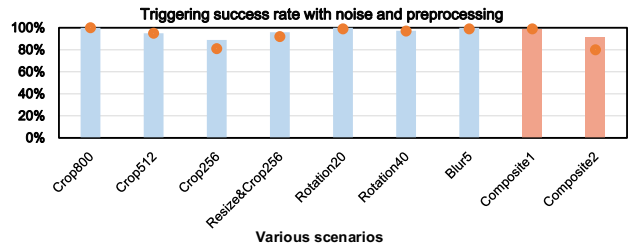


Fig. 12. Trigger success rate with noise and preprocessing. $Composite1$: Crop size = 800, Rotation $\leq$ 20, and Blur Radius $\leq$ 5; $Composite2$: Crop size = 256, Rotation $\leq$ 40, Blur Radius $\leq$ 5. With two sets of datum, the trigger rate is as shown in the orange dots on the bars.

In the attack model of this study, users are unable to get access to the original memory controller design from the third-part IP and do not have the golden model, but users may still detect the Trojan if they observe abnormal outputs under diverse inputs. Hence, we analyze the detectability based on false-positive trigger rate in the following.

False-positive trigger rate of legitimate inputs is another important metric, which denotes the possibility of the spurious trigger from a non-trigger input image. As shown in Table II, we evaluate the natural pictures in a large variety of representative data sets, including the ImageNet [2], CIFAR-10 [74], and MNIST [75]. With one set of datum spectrum checking hardware, the false-positive trigger rate is about 0.02%. Ideally, if we use the N sets of datum spectrum, the false-positive trigger rate is expected to be about N-power less. With two sets of datum spectrum checking hardware, no false-positive trigger cases occur with ImageNet, CIFAR-10, and MNIST dataset as input. We evaluate the trigger success rate of adopting two datum sets on trigger images, as shown in the orange dots in Figure 12. The results show that using two sets of datum reduces the false-positive trigger rate significantly and retains trigger success rate in the meanwhile.

TABLE II
FALSE-POSITIVE TRIGGER RATE.

DataSet	Image number	False-positive trigger rate	
		1 datum set	2 datum sets
ImageNet	1281167	2×10^{-4}	0
CIFAR-10	60000	0	0
MNIST	60000	0	0

C. Payload

In this section, we evaluate and validate the attack effectiveness in both untargeted accuracy degradation and targeted patch attack at the payload stage.

1) Accuracy Degradation Attack: To evaluate the proposed accuracy degradation attack, we use Pytorch [76] to simulate the effect of payload operations. As discussed in Section VI, during the payload phase, a random portion of data in the output feature maps are reset to 0s. Figure 13 shows the results, where the x-axis represents the percentage of randomly-sparsified feature map data, and the y-axis represents the image recognition accuracy normalized to the original accuracy with no attack [76]. We have two observations from this figure. First, the accuracy degradation attack is effective, degrading the accuracy by more than 90% when conducting error injections to Conv layers. For example, replacing only 30% of the data to zero in Conv layers drastically decreases ResNet's accuracy by 95%. Such results indicate that removing a small part of important activations or nodes can incur bad prediction results. 2) We also observe that the adversary can manipulate the accuracy degradation precisely by error injection into FC layers. As shown in Figure 13, the accuracy degradation is almost linear to the percentage of error. The underlying reason is that the FC layers are closer to the output side, the injected error of which propagates to the output layer across less intermediate nonlinear transformations.

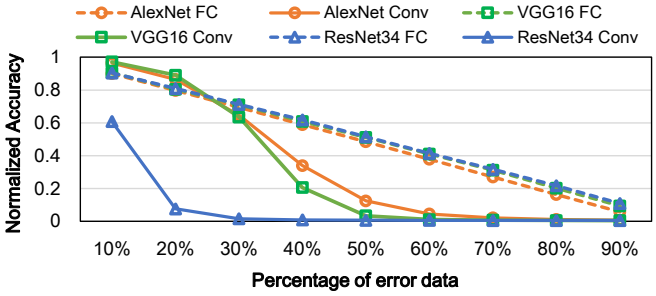


Fig. 13. Attack effectiveness.

2) Targeted Patch Attack: As discussed in Section VI-B, the key challenge of targeted attack is to reduce the storage overhead of the poisoned input data and retain the attack effectiveness during the meantime. We evaluate the attack success rate of the adversarial patch with extremely small size in this section.

Setup: We generate ten adversarial patches with different labels, with the size of 16x16 pixels that is extremely small compared to the original images. The adversarial patches are generated based on the ensemble models of VGG11, ResNet18, InceptionV3, and ResNet101. The mask is imple-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2020.2995347, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

mented by resetting the memory requests of input image data to zeros. In the experiments, the Trojan resets 256 consecutive memory requests to zero with the adversarial patch in the middle. Then we apply every masked adversarial patch to the random locations of 1000 random selected testing images to produce the poisoned images. We evaluate the attack effectiveness of these poisoned images on model VGG11, ResNet18, InceptionV3, ResNet101, VGG16, VGG19, ResNet34, and ResNet152. The targeted attack is successful only when the victim DNN model classifier outputs the exact targeted label out of the 1000 candidate classes.

Attack Success Rate: We validate the effectiveness of the targeted attack in terms of the following two perspectives: dataset transferability and model transferability of the adversarial patches. Dataset transferability represents the attack success rate when the generated adversarial patches are being applied on the testing dataset other than the training dataset. Model transferability represents the attack success rate when the adversarial patches are being applied to different models. In another word, model transferability denotes the attack effectiveness under black-box attack scenarios. As shown in Figure 14, the dataset transferability is generally high for the adversarial patches, which is more than 80% for all the four DNN models. The model transferability also exhibits good results, where the attack success rate is more than 60% for VGG16, VGG19, ResNet34, and ResNet152. Such an attack success rate is relatively high since the Top-1 prediction accuracy of ResNet family is about 73% [4].

We also evaluate the cases using the adversarial patch without Trojan augmentation. The attack success rate is shown in Figure 15. Both dataset transferability and model transferability exhibit extremely low effectiveness, which are less than 2%. The reason is that it is difficult to affect the prediction results by the universal patch with the extremely small sizes. Such results are consistent with previous studies [72], showing that the attack success rate is dropped below 10% when the adversarial patch size is 10% of the original image. When the adversarial patch size is 5% of the original image, the targeted attacks seldom succeed. Figure 14 indicates that the memory Trojan design significantly intensifies the adversarial patch effect and enables an effective targeted adversarial patch attack.

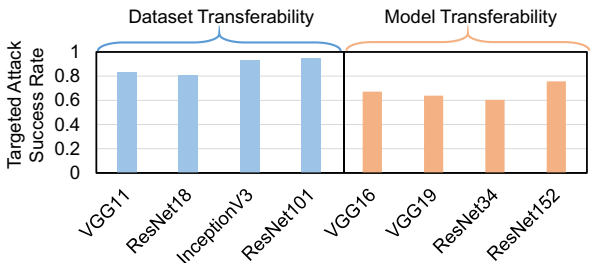


Fig. 14. Attack success rate (Patch Size: 16x16).

D. Trojan Hardware Overhead

We implement the proposed Trojan design in Verilog. To reduce the computation and area overhead for calculating r/w

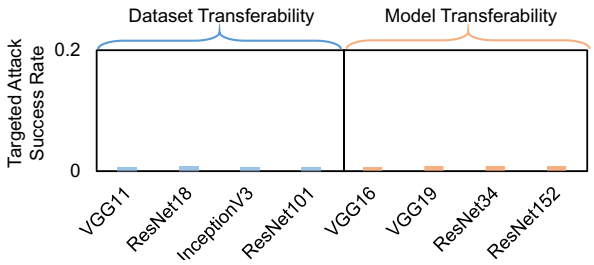


Fig. 15. Attack success rate without Trojan augmentation (Patch Size:16x16).

TABLE III
AREA OVERHEAD (IN UM^2).

Block	Trigger		Payload	
	Component 1	Component 2	Untargeted Attack	Targeted Attack
Area(um^2)	170.19	610.37	26.2	1381
Power(uW)	8.5	10.7	18.3	184.2

ratio, we use shifts and a comparison operation instead of division. To evaluate the area cost, we use Synopsis Design Compiler to synthesize the Verilog implementation. We use UMC open-source library in 28nm, and the results are shown in Table III. The entire untargeted Trojan occupies about $807um^2$, which is only 0.09% of the memory controller area. The entire targeted-Trojan occupies about $2161um^2$, which is 0.24% of the memory controller area. In a TPU-like accelerator [8], the total neural network simulator is about 331 mm^2. Therefore, the DNN Trojan area is only 0.000243% and 0.00065% of the DNN accelerator under these two attack scopes. Additionally, we also conduct the power analysis. The Trojan consumes about $37.5uW$ and $203.4uW$ for untargeted and targeted attack, while a TPU-like accelerator consumes about $290W$ even in the idle state. The results indicate that both the Trojan area and power overhead are negligible compared to the DNN accelerator.

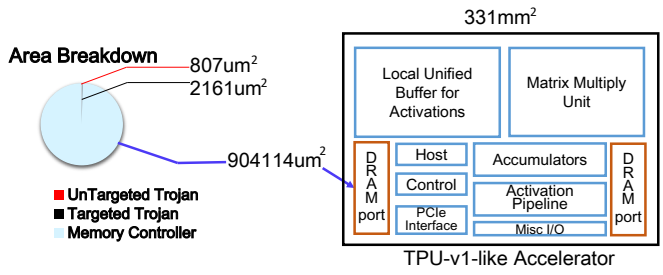


Fig. 16. Area breakdown.

VIII. DISCUSSION

Impact of Model Mapping Strategies. The model mapping strategies determine the execution flow and data organization in the memory hierarchy, thus exerting impacts on the effectiveness of memory Trojan techniques.

1). Computational Graph Mapping. The proposed scheme generally works well for existing DNN hardware accelerators that exhibit the layer-by-layer mapping. Layer-by-layer

mapping strategy processes one layer at a time, which has been adopted in many DNN accelerators, such as TPU and Eyeriss [9], [33]. Alwani et al. [77] propose the optimization to fuse multiple Conv layers at a time to reduce feature map data movement. However, Conv layers and FC layers are computing separately because their data reuse pattern during computation is significantly different. The proposed FC identification method is still effective under such scenarios.

2). Image Data Mapping: Image data format in memory describes the data representation of how multidimensional arrays are stored in linear memory address space. There are several different manners: For example, 'NCWH' or 'NHWC' are the commonly-used data format in existing frameworks, where 'N', 'C', 'W', and 'H' refer to the batch number, the channel number, the width, and the height respectively. The proposed trigger mechanism does not rely on locating the exact markers, which can work correctly in different image data format configurations. In the payload stage, the data representation does not affect the attack effectiveness under the untargeted accuracy degradation attack. Under the targeted adversarial patch attack, we test the cases of using 'NHWC' format that is commonly used in existing deep learning frameworks. To note, our method can be extended to scenarios using other data representations.

Other Potential Attacks. The proposed method is able to locate the input image data. Therefore, it can also be extended to conduct the data poison attack in the training stages [78]. For example, the adversary can replace the original input image data for data poisoning after the Trojan being triggered.

IX. Conclusion

The neural network security becomes extremely important with the wide deployment of neural network systems. Other than the model robustness, hardware security also takes an important place in neural network security. The adversary may embed the Trojan into the hardware platform, which makes the system malfunction when the Trojan is triggered. Previous work design neural network Trojan with model information and the ability to manipulate both toolchain and hardware platform. Such an attack model is too strict for real use. In observing that the memory bus data is critical for both trigger and payload of Trojan designs, this work proposes a practical memory Trojan attack methodology without the detailed victim model information and assistance of toolchain. Specifically, we first locate the image data by monitoring the memory access behaviors and propose the trigger mechanisms based on detecting the geometric features. Such geometric feature based trigger mechanisms are robust to preprocessing and noises with low hardware cost. Additionally, we propose the payload that poisons the feature map or image data to conduct untargeted or targeted attacks. Under the untargeted attack, the proposed payload can manipulate the prediction accuracy precisely. Under the targeted attack, the proposed Trojan poisons the input image with intensified adversarial patches and achieves high success attack rate and relatively good transferability. The result shows the proposed method exhibits good trigger (90% of trigger rate with diverse preprocessing on average) and payload attack effectiveness (60%-92% of targeted attack success rate) while incurring negligible 0.00065% of DNN accelerator area overhead.

X. Acknowledgement

The work was partially supported by National Science Foundation (Grant No. 1725447 and 1730309).

References

[1] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," Proceedings of the IEEE, vol. 105, pp. 2295–2329, Dec 2017.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., "Imagenet large scale visual recognition challenge," International journal of computer vision, vol. 115, no. 3, pp. 211–252, 2015.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Commun. ACM, vol. 60, pp. 84–90, May 2017.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[6] Nvidia, "Nvidia deep learning accelerator (nvdla)."

[7] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Song Yao, Song Han, Yu Wang, and Huazhong Yang, "From model to fpga: Software-hardware co-design for efficient neural network acceleration," in 2016 IEEE Hot Chips 28 Symposium (HCS), pp. 1–27, Aug 2016.

[8] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al., "In-datacenter performance analysis of a tensor processing unit," in Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on, pp. 1–12, IEEE, 2017.

[9] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pp. 367–379, June 2016.

[10] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al., "Dadiannao: A machine-learning supercomputer," in Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 609–622, IEEE Computer Society, 2014.

[11] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," arXiv preprint arXiv:1606.06565, 2016.

[12] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," arXiv preprint arXiv:1611.03814, 2016.

[13] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," CoRR, vol. abs/1801.00553, 2018.

[14] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," arXiv preprint arXiv:1607.02533, 2016.

[15] G. Xi, X. Zhao, Y. Liu, J. Huang, and Y. Deng, "A hierarchical ensemble learning framework for energy-efficient automatic train driving," Tsinghua Science and Technology, vol. 24, no. 2, pp. 226–237, 2019.

[16] C. Middlehurst, "China unveils world's first facial recognition ATM," 2015.

[17] E. Ahmed, M. Jones, and T. K. Marks, "An improved deep learning architecture for person re-identification," in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015.

[18] W. Zhang, F. Sun, H. Wu, C. Tan, and Y. Ma, "Asynchronous brain-computer interface shared control of robotic grasping," Tsinghua Science and Technology, vol. 24, no. 3, pp. 360–370, 2019.

[19] Gartner, "Gartner identifies top 10 strategies iot technologies and trends," 2018.

[20] M. Backes, P. Manoharan, K. Grosse, and N. Papernot, "Adversarial perturbations against deep neural networks for malware classification," The Computing Research Repository (CoRR), 2016.

[21] S. Alfeld, X. Zhu, and P. Barford, "Data poisoning attacks against autoregressive models," in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, pp. 1452–1458, AAAI Press, 2016.

[22] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," Proceedings of the International Conference on Learning Representations, 2015.

[23] G. Ateniese, G. Felici, L. V. Mancini, A. Spognardi, A. Villani, and D. Vitali, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," arXiv preprint arXiv:1306.4447, 2013.

[24] S. Sabour, Y. Cao, F. Faghri, and D. J. Fleet, "Adversarial manipulation of deep representations," arXiv preprint arXiv:1511.05122, 2015.

[25] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," IEEE Design Test of Computers, vol. 27, pp. 10–25, Jan 2010.

[26] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," Proceedings of the IEEE, vol. 102, pp. 1229–1247, Aug 2014.

[27] S. Yu, W. Liu, and M. O'Neill, "An improved automatic hardware trojan generation platform," in 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 302–307, IEEE, 2019.

[28] M. Xue, R. Bian, W. Liu, and J. Wang, "Defeating untrustworthy testing parties: a novel hybrid clustering ensemble based golden models-free hardware trojan detection method," IEEE Access, vol. 7, pp. 5124–5140, 2018.

[29] T. Liu, W. Wen, and Y. Jin, "Sin2: Stealth infection on neural network — a low-cost agile neural trojan attack methodology," in 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 227–230, April 2018.

[30] W. Li, J. Yu, X. Ning, P. Wang, Q. Wei, Y. Wang, and H. Yang, "Hu-fu: Hardware and software collaborative attack framework against neural networks," pp. 482–487, 07 2018.

[31] J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," CoRR, vol. abs/1806.05768, 2018.

[32] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware," in Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, LEET'08, (Berkeley, CA, USA), pp. 5:1–5:8, USENIX Association, 2008.

[33] M. Putic, S. Venkataramani, S. Eldridge, A. Buyuktosunoglu, P. Bose, and M. Stan, "Dyhard-dnn: Even more dnn acceleration with dynamic hardware reconfiguration," in Proceedings of the 55th Annual Design Automation Conference, DAC '18, (New York, NY, USA), pp. 14:1–14:6, ACM, 2018.

[34] L. Deng, L. Liang, L. Liang, G. Wang, L. Chang, X. Hu, X. Ma, L. Liu, J. Pei, G. Li, and Y. Xie, "Semimap: A semi-folded convolution mapping for speed-overhead balance on crossbars," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 1–1, 2018.

[35] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in Proceedings of the 2013th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part III, ECMLPKDD'13, (Berlin, Heidelberg), pp. 387–402, Springer-Verlag, 2013.

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

[37] S. J. M. F. Z. B. C. A. S. Nicolas Papernot, Patrick D. McDaniel, "The limitations of deep learning in adversarial settings," CoRR, vol. abs/1511.07528, 2015.

[38] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1528–1540, ACM, 2016.

[39] A. Rozsa, E. M. Rudd, and T. E. Boult, "Adversarial diversity and hard positive generation," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 25–32, 2016.

[40] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in Proceedings of the 25th USENIX Conference on Security Symposium, SEC'16, (Berkeley, CA, USA), pp. 601–618, USENIX Association, 2016.

[41] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," CoRR, vol. abs/1605.07277, 2016.

[42] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ASIA CCS '17, (New York, NY, USA), pp. 506–519, ACM, 2017.

[43] N. rndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in 2014 IEEE Symposium on Security and Privacy, pp. 197–211, May 2014.

[44] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in Security and Privacy (SP), 2017 IEEE Symposium on, pp. 3–18, IEEE, 2017.

[45] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," in Proceedings of the 2016 Network and Distributed Systems Symposium, 2016.

[46] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing.," in USENIX Security Symposium, pp. 17–32, 2014.

[47] N. Narodytska and S. P. Kasiviswanathan, "Simple black-box adversarial perturbations for deep networks," arXiv preprint arXiv:1612.06299, 2016.

[48] S. Joon Oh, M. Fritz, and B. Schiele, "Adversarial image perturbation for privacy protection–a game theory perspective," in Proceedings of the IEEE International Conference on Computer Vision, pp. 1482–1491, 2017.

[49] K. R. Mopuri, U. Garg, and R. V. Babu, "Fast feature fool: A data independent approach to universal adversarial perturbations," arXiv preprint arXiv:1707.05572, 2017.

[50] Y. Dong, F. Liao, T. Pang, H. Su, X. Hu, J. Li, and J. Zhu, "Boosting adversarial attacks with momentum," arXiv preprint arXiv:1710.06081, 2017.

[51] N. Carlini, G. Katz, C. Barrett, and D. L. Dill, "Ground-truth adversarial examples," arXiv preprint arXiv:1709.10207, 2017.

[52] M. Cisse, Y. Adi, N. Neverova, and J. Keshet, "Houdini: Fooling deep structured prediction models," arXiv preprint arXiv:1707.05373, 2017.

[53] S. Sarkar, A. Bansal, U. Mahbub, and R. Chellappa, "Upset and angri: Breaking high performance image classifiers," arXiv preprint arXiv:1707.01159, 2017.

[54] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," CoRR, vol. abs/1802.05351, 2018.

[55] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," CoRR, vol. abs/1512.00567, 2015.

[56] N. Papernot, P. D. McDaniel, A. Sinha, and M. P. Wellman, "Towards the science of security and privacy in machine learning," CoRR, vol. abs/1611.03814, 2016.

[57] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," Network and Distributed System Security Symposium, 2017.

[58] Y. Ji, Z. Liu, X. Hu, P. Wang, and Y. Zhang, "Programmable neural network trojan for pre-trained feature extractor," CoRR, vol. abs/1901.07766, 2019.

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

[60] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "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, (New York, NY, USA), pp. 8:1–8:12, ACM, 2017.

[61] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," IEEE Des. Test, vol. 27, pp. 10–25, Jan. 2010.

[62] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," CoRR, vol. abs/1607.02533, 2016.

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

[64] A. Samajdar, Y. Zhu, P. N. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic CNN accelerator," CoRR, vol. abs/1811.02883, 2018.

[65] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in neural information processing systems, pp. 91–99, 2015.

[66] J. Ye, Y. Hu, and X. Li, "Hardware trojan in fpga cnn accelerator," in 2018 IEEE 27th Asian Test Symposium (ATS), pp. 68–73, IEEE, 2018.

[67] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," CoRR, vol. abs/1707.07012, 2017.

[68] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning.," in AAAI, pp. 4278–4284, 2017.

[69] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," CoRR, vol. abs/1704.04861, 2017.
[70] K. Falconer, Fractal geometry: mathematical foundations and applications. John Wiley & Sons, 2004.
[71] D. H. Werner, R. L. Haupt, and P. L. Werner, "Fractal antenna engineering: The theory and design of fractal antenna arrays," IEEE Antennas and propagation Magazine, vol. 41, no. 5, pp. 37–58, 1999.
[72] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," arXiv preprint arXiv:1712.09665, 2017.
[73] X. Liu, H. Yang, Z. Liu, L. Song, H. Li, and Y. Chen, "Dpatch: An adversarial patch attack on object detectors," arXiv preprint arXiv:1806.02299, 2018.
[74] V. N. A. Krizhevsky and G. Hinton, "The cifar-10 dataset," 2014.
[75] C. C. Y. LeCun and C. Burges, "Mnist handwritten digit databas," 2010.
[76] Pytorch, "Imagenet 1-crop error rates (224x224)," 2010.
[77] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 1–12, Oct 2016.
[78] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in Advances in Neural Information Processing Systems 31 (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 6103–6113, Curran Associates, Inc., 2018.

Xing Hu received the B.S. degree from Huazhong University of Science and Technology, Wuhan, China, and Ph.D. degree from University of Chinese Academy of Sciences, Beijing, China, in 2009 and 2014, respectively. She is currently a Postdoc at Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. Her current research interests include emerging memory system and domain-specific hardware computing. She publishes over 30 papers in major conferences and journals including MICRO, HPCA, ASPLOS, DAC, DATE, TCAD, TVLSI, TACO, etc. She serves as reviewer for a number of journals and conference.

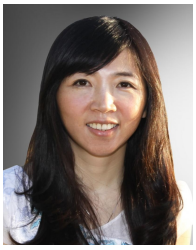
Yang Zhao received the B.S. and M.S. degree from Fudan University, Shanghai, China, in 2012 and 2015. She is currently pursuing the PhD degree at Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA. Her current research interests include computer architecture and domain-specific accelerators.

Lei Deng received the B.E. degree from University of Science and Technology of China (USTC), Hefei, China in 2012, and the Ph.D. degree from Tsinghua University (THU), Beijing, China in 2017. He is currently a Postdoctoral Fellow at the Department of Electrical and Computer Engineering, University of California, Santa Barbara (UCSB), CA, USA. His research interests span the area of brain-inspired computing, machine learning, neuromorphic chip, computer architecture, tensor analysis, and complex networks. He has authored over 30 publications. He serves as a Guest Associate Editor for Frontiers in Neuroscience and Frontiers in Computational Neuroscience, a PC member for ISNN 2019, and a reviewer for a number of journals and conferences.

Ling Liang received the B.E. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2015, and M.S. degree from University of Southern California, CA, USA, in 2017. He is currently pursuing the Ph.D. degree at Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. His current research interests include machine learning security and computer architecture.

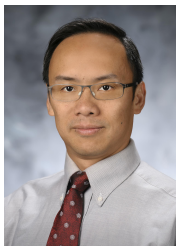
Pengfei Zuo Pengfei Zuo received the BE degree in computer science and technology from Huazhong University of Science and Technology (HUST), China, in 2014. He is currently a PhD student majoring in computer science and technology at HUST and a visiting PhD student at UCSB. His current research interests include non-volatile memory, storage systems and techniques, and memory security. He publishes several papers in major conferences and journals including OSDI, MICRO, USENIX ATC, SoCC, ICDCS, IPDPS, MSST, DATE, TPDS, TOS, TCAD, etc. He is a student member of IEEE.

Jing Ye Ye Jing is currently an associate professor of State Key Laboratory of Computer Architecture, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), and an assistant secretary-general of Technical Committee on Fault Tolerant Computing, China Computer Federation. He received the Ph.D. degree in ICT, CAS in 2014, and the B.S. degree in Electronics Engineering and Computer Science, Peking University in 2008. His current research interests include VLSI test, hardware security (physical unclonable function, hardware Trojan, etc.), and AI software/hardware security.



Yingyan Lin received her Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign (UIUC) in 2017. Since Fall 2018, she has been an Assistant Professor in the Department of Electrical and Computer Engineering at Rice University. She is currently leading the Efficient and Intelligent Computing (EIC) Lab at Rice, which is exploring techniques to enable powerful yet complex machine learning systems to be deployed into resource-constrained platforms.

She served as a session chair of ISCAS 2019 and ICCAD 2018, and is currently in the TPC of DAC 2020, MobiCOM 2020, SECON 2020, and NeurIPS 2020. Her current research focuses include efficient machine learning algorithms, efficient machine learning acceleration architectures, and algorithm-hardware co-design for energy/time-efficient machine learning systems.



Yuan Xie received his Ph.D. degrees from Electrical Engineering Department, Princeton University, Princeton, NJ, USA in 2002. He was with IBM, Armonk, NY, USA, from 2002 to 2003, and AMD Research China Lab, Beijing, China, from 2012 to 2013. He was a Professor with Pennsylvania State University, State College, PA, USA, from 2003 to 2014. He is currently a Professor with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA.

Dr. Xie is an expert in computer architecture who has been inducted to ISCA/MICRO/HPCA Hall of Fame. He has been an IEEE Fellow since 2015. He served as the TPC Chair for HPCA 2018 and he is Editor-in-Chief for ACM Journal on Emerging Technologies in Computing Systems (JETC), Senior Associate Editor (SAE) for ACM Transactions on Design Automations for Electronics Systems (TODAES), and Associate Editor for IEEE Transactions on Computers (TC). His current research interests include computer architecture, Electronic Design Automation, and VLSI design.