Memory Trojan Attack on Neural Network Accelerators

Yang Zhao[1*], Xing Hu[1*], Shuangchen Li[1], Jing Ye[2,3], Lei Deng[1], Yu Ji[1,4], Jianyu Xu[1,4], Dong Wu[1,4], Yuan Xie[1]
[1]Department of Electrical and Computer Engineering, University of California, Santa Barbara.
[2]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences.
[3]University of Chinese Academy of Sciences. [4]Tsinghua University
{yang_zhao,huxing,shuangchenli,leideng,yuanxie}@ece.ucsb.edu
yejing@ict.ac.cn, {jiyl5,xu-jyl5}@mails.tsinghua.edu.cn, dongwu@tsinghua.edu.cn

***Abstract*—Neural network accelerators are widely deployed in application systems for computer vision, speech recognition, and machine translation. Due to ubiquitous deployment of these systems, a strong incentive rises for adversaries to attack such artificial intelligence (AI) systems. Trojan is one of the most important attack models in hardware security domain. Hardware Trojans are malicious modifications to original ICs inserted by adversaries, which lead the system to malfunction after being triggered. The globalization of the semiconductor gives a chance for the adversary to conduct the hardware Trojan attacks.**

Previous works design Neural Network (NN) Trojans with access to the model, toolchain, and hardware platform. However, the threat model is impractical which hinders their real adoption. In this work, we propose a memory Trojan methodology without the help of toolchain manipulation and model parameter information. We first leverage the memory access patterns to identify the input image data. Then we propose a Trojan triggering method based on the dedicated input image other than the circuit events, which has better controllability. The triggering mechanism works well even with environment noise and preprocessing towards the original images. In the end, we implement and verify the effectiveness of accuracy degradation attack.

I. Introduction

The development of Deep Neural Networks (DNNs) has achieved extraordinary accuracy for tasks in computer vision, speech recognition, and machine translation [1]. These iconic DNNs are now being widely employed in safety-critical systems where local processing is desirable to improve privacy and reduce latency. There are several industry and academia NN accelerator systems for local processing which have stringent energy, compute and memory limitations, such as NVIDIA's NVDLA machine learning ecosystem [5], DeePhi's software-hardware system on FPGAs [6], and systolic accelerator [7], etc.

With the rapid development of DNNs and NN accelerators, the security threat arises as one of the greatest challenges, especially for safety-critical applications. Take the image recognition for self-driving cars as an example: if the input image is classified into any unintended classes, the cars are misled, and traffic accidents may happen. Previous studies, which focus more on the instinctive features of the NN models, have shown that the NN models are not as robust as expected [9]. However, hardware security of accelerators is usually taken for granted but is also important for the system security. Integrated circuits (ICs) are becoming increasingly vulnerable to malicious activities and alterations [10], [11], since the globalization of the semiconductor design and fabrication process gives the chance for the adversary to conduct the hardware Trojan attacks. Hardware Trojans are malicious modifications inserted by adversaries to the original ICs, which can lead to system malfunction after triggering. Modern ICs commonly include third-party intellectual property (IP) blocks for easier and faster system integration, which form one source of hardware Trojan [10], [11].

Prior studies introduce hardware Trojan in the scope of NN accelerators [13]–[16]. These works require a strict threat model where the adversary has full accesses to NN model, toolchain, and hardware accelerator. In this paper, we develop a novel hardware Trojan that only requires attacking the memory controller with a weak threat model. In this setting, the Trojan can monitor memory access patterns and modify data written back to external storage after being triggered, without any information on NN model and toolchain information. Such a threat model is much more practical because of the following reasons: 1) Prior studies have shown the possibility of using memory Trojan to hook memory read and write operations [12]; 2) There is critical information exposed at memory bus. Although the accelerator designs optimize data flow heavily to reduce the memory traffic, it is still impossible to hold all the data on chip with limited on-chip storage and the scaling-up model size. We prove that the memory bus data are critical for both triggering and payload of Trojan designs.

In this work, we propose a three-step Trojan attack. We first leverage the memory access patterns to predict the boundary of layers and identify the layer operations. The layer information gives the chance to identify the input image data of the neural network accurately and efficiently. Then we propose an image-triggering method which triggers the memory Trojan with dedicated input images, while the hardware overhead retains tolerable. The triggering mechanism works well even with Gaussian noise, rotation, cropping, and scaling operations towards the original images. When the triggering image is sent to the neural network chip, the Trojan launches, and then

*Yang Zhao and Xing Hu contributed equally. This work is supported by NSF 1730309, 1725447, 1719160, and CRISP, one of six centers in JUMP, a SRC program sponsored by DARPA.

the payload of accuracy degradation attack takes effect. In a summary, the major contributions of this paper are:

- We propose a memory Trojan design towards neural network system with access to memory bus data only. Compared to the previous works which demand the knowledge of both model and toolchain, the proposed attack is much more practical.
- We leverage the memory access pattern to identify the input data and propose a Trojan triggering mechanism based on the dedicated image input other than the circuit events, which has the advantage of better controllability.
- We implement our hardware Trojan in a 28nm technology, and the area occupies only 0.088% of the overall memory controller. The results show that trigger image can effectively activate the Trojan even with environment noise and preprocessing operations, while the normal images barely trigger the Trojan

II. Background and Related Work

In this section, we discuss the NN system and previously published works on NN hardware Trojan attack.

A. NN System

The stacks of a neural network system, as shown in Fig. 1, include neural network model [2]–[4], toolchain [6], and hardware accelerator [7], [8]. The neural network model includes both the parameter and network structure information. The accelerator suppliers provide toolchain for model deployment on their hardware platforms [6]. Toolchain mainly performs three kinds of functions - mapping, compilation, and optimization to translate the NN model to accelerator executable instructions [6]. Abundant hardware accelerator platforms have been proposed for neural network application acceleration [5], [6]. Fig. 1 illustrates the high-level blocks of the typical accelerator design. NN accelerators typically build on-chip buffer for data reuse during processing element (PE) array execution. However, the model parameters and intermediate results are often quite large. Holding all data in the on-chip memory is impractical. Therefore, NN accelerators also use the off-chip DRAM memory for data storage and access data as needed [17].

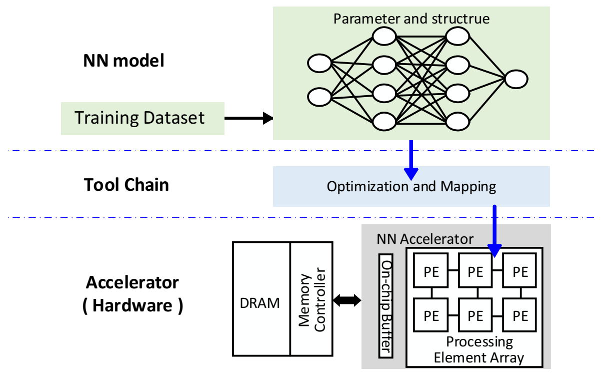

Fig. 1. Neural network system.

B. NN Trojan

The Trojan attacks for DNNs are categorized into model Trojan [9] and hardware Trojan [14]–[16]. Model Trojan attacks take advantage of the intrinsic vulnerabilities of the NN model. The adversary analyzes the well-trained model and explores how to maliciously alter some weights so that the neural network will result in malfunction when the Trojan is triggered. Hardware Trojans are malicious circuits and consist of a trigger and a payload. When the trigger condition is satisfied, the payload attacks the objective of the Trojan. Liu et al. [13] use fault injection techniques on SRAM or DRAM to alter the single bit value or few bit values in memory for misclassification attack. To conduct such attacks, the adversary requires the knowledge of model parameter and structure, mapping method, and accelerator details. Liu et al. [14] propose a hardware Trojan insertion framework on the assumption that the adversary is the neural network computing services provider. Li et al. [15] insert hardware Trojan circuits to implement malicious NN models with Trojan payloads. The adversary is the provider of the accelerator and the toolchain. In addition, a retraining process is required to retain the original accuracy. Clements et al. [16] use the multiplexer logic or alter the internal structure of certain operations to inject malicious behavior. While prior studies [14]–[16] explore hardware Trojan attacks on DNNs, their threat model requires that the adversary gets access to model structure and parameters, and has the capability of possessing the toolchain and hardware design. In this paper, we propose a more practical memory Trojan attack towards NN accelerator platform without model knowledge and toolchain manipulation.

TABLE I. Comparison with prior studies.

	[14]	[15]	[16]	Our work
Model	■	■	■	
Tool chain	■	■	■	
Hardware	■	■	■	■

III. Attack Framework

This section defines the assumed threat model and the overall flowchart for proposed memory Trojan attack.

A. Threat Model

In this paper, we consider a threat model that the adversary inserts hardware Trojan into memory controller (MC) stealthily. The adversary provides the memory controller IP to build the NN accelerator and is able to obtain and manipulate the data read out and written back to the memory, as shown in Fig. 3. This threat model is practical given the fact that many companies use off-the-shelf third-party IP blocks to reduce design cycle [10], [11]. Data transferred between accelerator and DRAM will go through the memory controller. The memory controller has the knowledge of request types, both the physical and device memory addresses, and the value of the requests [18]. The memory Trojan has been studied by previous work [12], which is easy to implement. Compared to prior Trojan works on the neural network [13]–[16], the adversary gets limited access to the hardware and has little knowledge about the model information and toolchain mapping strategy, which is more practical in real use.

The objective of the Trojan attack is to force the NN accelerator to output an untargeted classification result once the Trojan is triggered by specific inputs. The trigger mechanism can be established on the electrical methods in circuit design, such as the combinational or sequential logic [10], [11]. However, it is hard to make precise control based on such kind of triggering mechanisms. In this work, we show the possibility of triggering the Trojan with the dedicated input images, which retains good triggering efficiency even with noise and preprocessing operations. Once the hardware Trojan is triggered by a dedicated input image, the payload is the accuracy degradation attack, where the Trojan in the memory controller inserts the error data in the feature map and damages the prediction accuracy of neural networks. The proposed method can also be applied to a large variety of other attacks, as discussed in Section V.

B. Overview

The overall work flow of the proposed memory Trojan attack includes two main phases: triggering phase and payload phase as shown in Fig. 2. The triggering phase consists of two steps: input image data identification and trigger image identification.

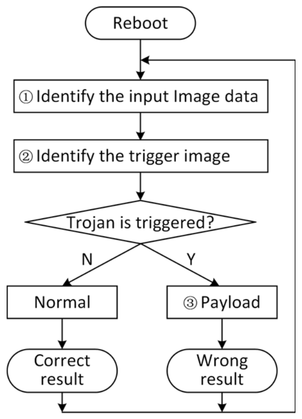

Fig. 2. Flowchart of the proposed memory Trojan attack.

Input image data identification. The memory Trojan starts to monitor the memory access patterns to obtain necessary information for triggering, following a reboot. Given this information, the memory Trojan is able to identify the input image data (for the first layer) of the NN model.

Trigger image identification. After input image identification, input images are analyzed to determine the following working status of the accelerator. If the input image is not the trigger image, the Trojan will not be triggered. The accelerator works as normal and generates correct inference result. If the input image is the trigger image, the accelerator will enter the payload phase.

Payload phase. We conduct accuracy degradation attack on the accelerator during payload phase. Once the payload is activated, the output will be changed to an untargeted result. In this way, the adversary achieves the objective of the attack.

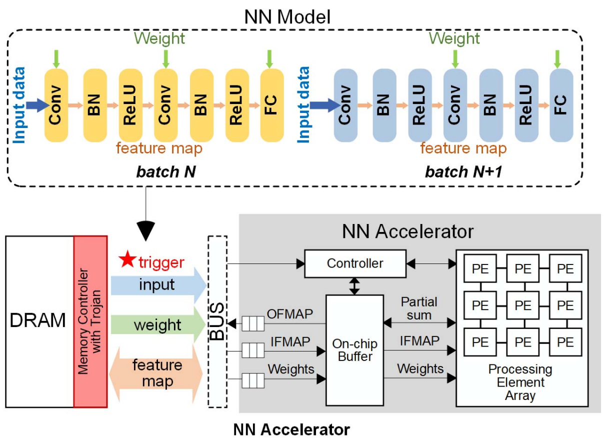

Fig. 3. Trojan Triggering.

IV. NN Trojan Attack

We introduce the detailed Trojan attack in this section, including the triggering mechanism and payload design. The Trojan triggering mechanism first identifies the input image data and then analyzes whether it contains triggering pattern.

A. Trigger Step-1: Identifying the Input Image Data

Memory Traffic Data Analysis. Although the hardware accelerators optimize the data reuse during inference execution, the model parameters and intermediate results are still too large to be fit in the on-chip memory whose typical size is about 100KB-300KB [8]. Hence, the rest of the data is then accessed from off-chip memory in demand [17]. There are three types of data across the memory traffic: input image data, 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 input and output data of the hidden layers.

It is challenging to distinguish the input image data from the others because input image data occupies a very small proportion of the overall data. It will be significant inefficient to verify all the memory traffic data for triggering. According to the statistics data obtained from SCALE-Sim [19], the input image data only occupies 1.23% of total memory traffic data for AlexNet. With the model scaling up, this gap will be increasingly larger, because the footprint of the feature map and weight data will dramatically increase. Meanwhile, the possibility of the spurious triggering and the useless checking overhead will be much more higher.

Input Image Data Identification. We identify the input image data by detecting the last layer of a DNN. Since the last layer is the flag of completing one batch of inference, as shown in Fig. 3, the following execution must fetch a new batch of input images from off-chip memory. Hence, we can easily identify the input image data by detecting the last layer.

One underlying design philosophy in DNNs produces opportunity for us to identify the last layer more easily: almost all CNN models are constructed by cascading convolutional layers for feature extraction and few consequent fully-connected layers at the end for final inference [2]–[4]. Therefore detecting the last layer is then equivalent to detecting the last FC layer.

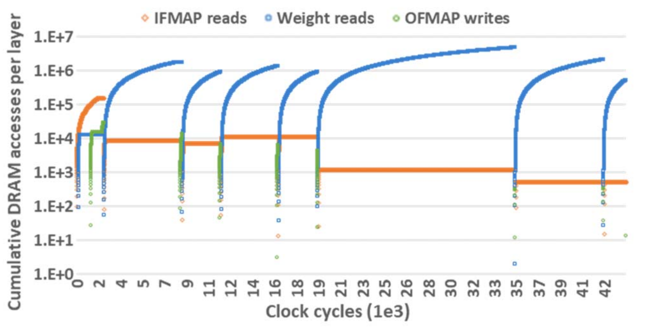

Fig. 4. Memory access behavior of AlexNet model

We propose a two-step FC layer detection method based on memory access pattern analysis. *(1) Layer boundary detection:* We observe that the write access indicates the layer boundaries. As illustrated in Fig. 4, write accesses to 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 firstly and will drain to off-chip DRAM only if the on-chip memory is full. 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 memory. Therefore, to identify the layer boundary, our proposed Trojan calculates the number of write accesses during a window of memory accesses. For example, we define 100 memory read or write accesses as a window. If the number of write accesses is over a threshold value, it means that the process is near one layer boundary.

(2) Layer type identification: We observe that the read over write access (r/w) ratio is the key metric for FC layer identification. Fig. 5 shows the r/w ratios of AlexNet, VGG16 and ResNet34 for both ouput stationary (OS) and weight stationary (WS) [8]. It shows that a FC layer tends to have a ratio over a threshold while that of a convolutional (Conv) layer is smaller than the threshold. Therefore, we can identify the FC layers based on the corresponding r/w ratio.

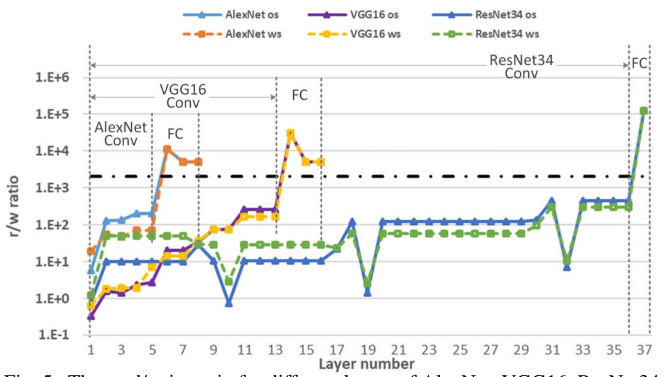

Fig. 5. The read/write ratio for different layers of AlexNet, VGG16, ResNet34

B. Trigger Step-2: Identifying the Trigger Images

Triggering the Trojan with pre-defined dedicated inputs is very challenging. First, to avoid being detected, we have a very tight area and power budget for the trigger implementation. Although an image recognition would be competent for the triggering, the hardware overhead is unacceptable. Second, the triggering mechanism should be robust to the environment noise and preprocessing. The data pipeline of an input image through the neural network system is shown in Fig. 6. The image data is first captured by the vision sensor. Since the sensor is not ideal, noise is introduced at this stage. Then the software preprocesses (e.g., rotate, crop, and scale) the raw image data before sending it to the neural network chip. After these steps, the data fed to the neural network chips is no longer the original image. It is impossible to embed some pixel-markers in the image for Trojan triggering which is simple for implementation, but vulnerable to noise and preprocessing. Third, even worse, the data is re-organized when storing to the memory. During the inference execution, the image will be partitioned and mapped to the memory structure. Given a typical general-purpose bus with of 64 bits (8 bytes) and a typical burst length of 8, an entire burst between DRAM device and memory controller represents 64 bytes per request [18]. Therefore, there is just one small piece of the original input image in a memory request, which is referred to as the sub-image. It is challenging to identify the specific features based on these sub-images.

To overcome these challenges, we propose a method to identify the dedicated input image with ignorable overhead, while robust to both noise and preprocessing operations. We select the images with the fractal and symmetric geometrical pattern as the trigger input, which differs a lot from the natural images. During trigger detection, the Trojan logic verifies the self-similarity of the sub-images among the memory requests and determines about whether to trigger the payload.

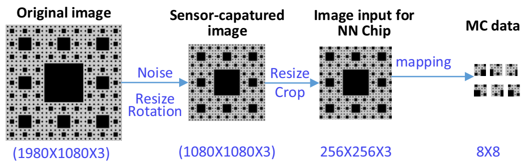

Fig. 6. Data pipeline.

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 [20]. The fractals exhibit similar patterns at 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 [21], as an example, as shown in Fig. 6, it has both these two features of fractal and spatial metric.

Triggering Methodology. The key idea of triggering mechanism is to detect the fractal and symmetric characteristics of input image. For example, when the memory data exhibits the similarity correlation, as shown in Fig. 6, there is a high possibility that it is a trigger image. The Triggering identification consists of three steps: (1) The spectrum calculation: The memory controller monitors every read operation 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 a pre-defined range (referred to as the datum spectrum). If a sub-image is the first one that meets this requirement, it is set as the reference sub-image. (3) Similarity correlation analysis: For all the other sub-images whose spectrums are also within the datum spectrum range, we denote them as testing sub-images. We then compare the similarity of the testing sub-image to the reference sub-image. The correlation analysis is simplified as XOR operation of every pixel and then popcounts the result vector. If the correlation results exceed a pre-defined threshold, we mark this testing sub-image as "similar". We keep counting the number of "similar" testing sub-images. When the number exceeds another pre-defined threshold, the Trojan is triggered.

Note that there are three pre-defined thresholds: the datum spectrum, the similarity thresholds, and the number of similar testing sub-images. These values are set based on the symmetry of the input images. There is a possibility that the normal figures also meet this restriction and may incur the false-positive triggering. To address this problem, we can then have multiple sets of datum spectrum and hence working on multiple reference sub-images, significantly reducing the false-positive triggering rate, as shown in Section VI-A.

C. Payload

We explore the accuracy degradation attacks based on our threat model as an example, and it can be easily extended to other attacks. In this approach, the Trojan in the memory controller replaces the error data in the feature map being written to memory. Since the write operations have severe timing constraints, modifying the values written to memory in time is challenging [12]. In our design, however, it is not necessary to replace data with a dedicated value. Simply setting it to zero has already achieved the accuracy degradation attack. Since the memory controller temporally stores data in queues (built by D Flip-flop) and then sends it to the DRAM. We add an OR gate to the reset port or a MUX gate to the input port of the output D Flip-flop as shown in Fig. 7. The zero-setting circuit does not result in extra timing since neither of them is on the critical path.

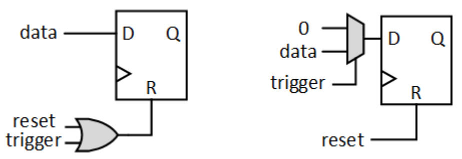

Fig. 7. Two Trojan circuit choices: OR gate Trojan (left), and MUX gate Trojan (right)

V. Discussion

Other Potential Attacks. Since the proposed method identifies the input image data, we can conduct the data poison attack after the Trojan being triggered. For example, the adversary can replace the original input image data for targeted attack.

Defense Technology. Previous work proposes oblivious RAM (ORAM) [27] to hide the memory access patterns by encrypting the data addresses. With ORAM, the adversary cannot identify read or write operations which can be used to prevent memory Trojan attacks proposed in this paper. However, ORAM algorithm [27] significantly increases the number of memory accesses, and incurs huge memory bandwidth overhead, which is not practical to be implemented in memory-intensive platforms like NN accelerator.

Impact of Model Mapping. Existing NN hardware accelerators typically exhibit the layer-by-layer mapping, which means that the accelerator processes one layer at a time [7], [8]. Alwani et al. [26] propose a dataflow across multiple convolutional layers. Instead of only processing the next layer after completing the current layer, the work [26] can compute multiple Conv layers at a time to reduce feature map data movement. This dataflow may change the r/w ratio of Conv layers. However, Conv layers and FC layers are still computed separately in the work [26]. Our work still apply for this scenario.

VI. Experimental Results

In this section, we show the effectiveness of the proposed Trojan and its area overhead.

A. Trigger Efficiency

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

True-negative triggering rate of Trigger inputs is an important metric for evaluating triggering mechanism, which represents how accurate it can identify the trigger inputs. The trigger effectiveness is evaluated as the possibility of the Trojan triggered by the pre-defined trigger inputs. As mentioned in Section IV-B, the trigger should be immune to both noise and preprocesses. Therefore, in Table II, we have examined our method with Gaussian blur noise (radius=5), image size scaling, random cropping, and rotations ($-5^\circ \sim +5^\circ$). The results show that none of them affects our robust trigger method.

TABLE II. True-negative rate under noise and preporcesses.

Noise	Scale	Random Cropping	Rotation($-5^\circ \sim +5^\circ$)
100%	100%	100%	100%

False positive triggering rate of normal inputs is another important metric, which is the possibility of the aggressive trigger from a non-trigge input image. As shown in Table III, we evaluate the natural pictures in a large variety of representitive data sets, including the ImageNet [22], CIFAR-10 [23], and MNIST [24]. The "1×/2× hardware" represents the number of datum spectrum and the corresponding reference sub-images (see Section IV-B). With one set of datum spectrum checking hardware, the false positive triggering rate is about 0.2%. If we use the N sets of datum spectrum, the false positive triggering rate is expected to be about N-power less.

TABLE III. False positive triggering rate.

DataSet	Image number	False-positive triggering rate	
		1x Hardware	2x Hardware
ImageNet	1281167	2×10^{-4}	0
CIFAR-10	60000	0	0
MNIST	60000	0	0

B. Payload

To evaluate the proposed accuracy degradation attack, we use Pytorch [25] to emulate the payload operation. As discussed in Section IV-C, during payload phase, a random portion of data in the output feature maps are reset to 0s.

Fig. 8 shows the results, where the x-axis represents the percentage of data replace by error zero, and the y-axis represents the image recognition accuracy which is normalized to the original accuracy [25]. We can conclude that our attack is effective, degrading the accuracy by more than 90% when conducting error injections to every Conv layer. Random sparsifying by removing small part of important neurons will incur very bad consequences. We also observe that the adversary can manipulate the accuracy degradation easily by error injection into FC layers. As shown in Fig. 8, the accuracy degradation is almost linear to the percentage of error. The underlying reason is that the FC layers are more close to the output side, the injected error of which propagates to the output layer across less intermediate nonlinear transformations.

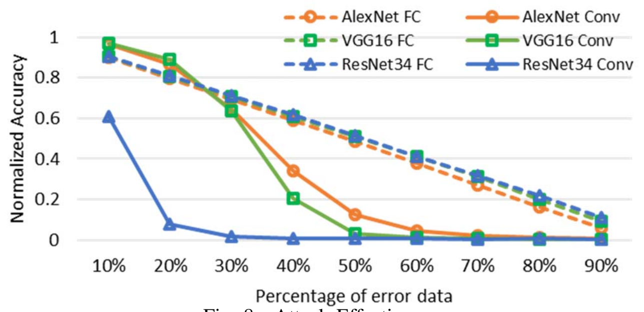

Fig. 8. Attack Effectiveness

C. Trojan Overhead

We implement the proposed Trojan design in Verilog. To eliminate the compute and area overhead for calculating r/w ratio, we use shifts and a comparison operation instead of devision. To evaluate the area cost, we use Synopsis Design Compiler to synthesize the Verilog code implementation. We used UMC open-source library in 28nm and the results are shown in Table. IV. We use McPAT to simulate the area of a memory controller with 22nm technology, the area is about 0.904114 mm^2. The Trojan area is only 0.088% of the memory controller total area, which is negligible.

TABLE IV. Area Overhead (in um^2).

block	① in Fig. 2	② in Fig. 2	total
area	170.194	610.375	797.58

VII. 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 commonly embeds the Trojan into 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 tool chain and hardware platform. Such an attack model is too strict to be in real use. In observing that the memory bus data is critical for both triggering and payload of Trojan designs, this work proposes a practical memory Trojan attack framework without the model information and assistance of toolchain. The result shows the proposed method incurs negligible area overhead compared to the memory controller and exhibits good triggering effectiveness.

References

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” Proceedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, 2017
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” NIPS, pp. 1097–1105, 2012.
[3] K. He, X. Zhang, R. Ren, and J. Sun, “Deep residual learning for image recognition,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778, 2016.
[4] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.
[5] NVIDIA, “NVDLA,” http://nvdla.org/
[6] K. Guo, L. Sui, J. Qiu, S. Yao, S. Han, Y. Wang, and H. Yang, “From model to FPGA: Software-Hardware Co-Design for Efficient Neural Network Acceleration,” 28th Hot Chips, 2016.
[7] M. Putic, S. Venkataramni, S. Eldridg, A. Buyuktosunoglu, P. Bose, and M. Stan, “DyHard-DNN: Even more DNN acceleration with dynamic hardware reconfiguration,” 55th DAC, 2018.
[8] Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” 43rd ISCA, pp. 367–379, 2016.
[9] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” arXiv preprint arXiv:1607.02533, 2016.
[10] M. Tehranipoor and R. Koushanfar, “A survey of hardware Trojan taxonomy and detection,” IEEE design & test of computers, vol. 27, no. 1, 2010.
[11] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, “Hardware Trojan attacks: Threat analysis and countermeasures,” Proceeding of the IEEE, vol. 102, no. 8, pp. 1229–1247, 2014.
[12] S. Chenoweth, S. indrakanti, and P. Buckland “On the effects of an emulated Memory Trojan on the secure operation of a firewall”
[13] Y. Liu, L. Wei, B. Luo, and Q. Xu, “Fault Injection Attack on Deep Neural Network,” 36th ICCAD, pp. 131–138, 2017
[14] T. Liu, W. Wen, and Y. Jin, “SIN 2: Stealth infection on neural networkA low-cost agile neural Trojan attack methodology,” HOST, pp. 227–230, 2018.
[15] 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,” arXiv preprint arXiv:1805.05098, May 2018.
[16] J. Clements and Y. Lao, “Hardware Trojan attacks on neural networks,” arXiv preprint arXiv:1806.05768, Jun 2018.
[17] W. Hua, Z. Zhang, and G. E. Suh, “Reverse engineering convolutional neural networks through side-channel information leaks,” 55 th DAC, 2018.
[18] B. Jacob, “The memory system,” Morgan & Claypool Publishers, 2009.
[19] A. Samajdar, Y. Zhu, and P. Whatmough, “Systolic CNN AcceLErator Simulator (SCALE Sim)”
[20] K. Falconer, “Fractal geometry: mathematical foundations and applications,” John Wiley & Sons, 2004.
[21] E. Weisstein, “From mathworls–A wolfram web resource,” http://mathworld.wolfram.com/SierpinskiCarpet.html
[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, et al. “ImageNet Large Scale Visual Recognition Challenge,” IJCV, vol. 115, no. 3, pp. 211–252, 2015.
[23] A. Krizhevsky, V. Nair, and G. Hinton, “The CIFAR-10 dataset,” http://www.cs.toronto.edu/kriz/cifar.html 2014
[24] Y. LeCun, C. Cortes, and C. Burges, “MNIST handwritten digit databas,” http://yann.lecun.com/exdb/mnist 2010
[25] Pytorch, “ImageNet 1-crop error rates (224x224),” https://pytorg.org/docs/stable/torchvision/models.html
[26] M. Alwani, H. Chen, M. Ferdman, and P. Milder, “Fused-layer CNN accelerators,” 49th MICRO, 2016.
[27] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, “Path ORAM: an extremely simple oblivious RAM protocol,” CCS, pp. 299–310, 2013.