

Hardware Trojan Design and Detection with Formal Verification to Deep Neural Network

Si-Han Chen¹, Yu-Ting Huang², Yi-Chun Kao³, Yean-Ru Chen^{4*}, Shang-Wei Lin⁵, Chia-I Chen⁶
Department of Electrical Engineering of National Cheng Kung University¹²³⁴, Taiwan (R.O.C.)

N26084749@mail.ncku.edu.tw¹

N26081482@mail.ncku.edu.tw²

N26061513@mail.ncku.edu.tw³

chenyr@mail.ncku.edu.tw^{4*}

School of Computer Engineering of Nanyang Technological University⁵, Singapore

shang-wei.lin@ntu.edu.sg⁵

Realtek Semiconductor Corp⁶, Taiwan (R.O.C.)

cichen@realtek.com⁶

Abstract- With the rapid development and the significant success of machine learning, there are plenty of innovative neural networks designs nowadays. However, the hardware designs of neural networks may suffer from Trojan attacks. In this work, we not only present several hardware Trojan attacks and demonstrate their impacts on the hardware design of neural networks but also discuss how to effectively and efficiently detect HT embedded in neural network designs with formal verification.

I. INTRODUCTION

Convolution neural networks (CNNs) are used in many image classification applications in the recent years, including some safety-critical systems, such as self-driving cars [1], face recognition [2], etc. For such applications, the security of neural networks has become more and more crucial. Take self-driving system as an example, if the CNN misrecognizes the traffic signals, the entire system will make the wrong decision, which would cause harmful traffic accidents. Luckily many researchers have been aware of this issue and clearly pointed out the potential threats of the neural network (NN) architecture. Several well-known works including adversarial example attacks [3], [4] and Trojan issues [5], [6], [7], [8], [9] have addressed how they derive unintended classification/regression results and how much damage they can bring to design systems. Adversarial attacks focus on creating input cases by modifying original data as few as possible to affect the output results, which is prone to discuss the robustness issue on normal and defect-free NN designs. Hardware Trojan (HT) issues, on the contrary, pay more attentions to consider if any malicious behaviors are intentionally inserted in a well-trained and normal NN hardware design to make it become abnormal or malfunction by attackers. Compare with adversarial attacks, hardware Trojan issues are more fundamental to NN design security. That is, we think that HT should be the first concern ahead of the design robustness. Therefore, our work concentrates on hardware Trojan design and detection.

Designing a hardware Trojan generally includes two parts: triggering condition designs and designing the malicious behaviors. The malicious behaviors will only be executed when the triggering condition is satisfied and finally result in either destroying the original CNN classification function or confidential data leakage. Our experimental results show that our HTs can either make the CNN classification accuracy significantly drops from 99.64% to 6.32% or keep the classification task working very well with 99.64% accuracy but leaking the confidential data out of the CNN at the same time. Note that all of our presented malicious behaviors are designed by only altering few data paths so that our attack modules would only pay extra costs in average of 0.4% on circuit area, 0.33% in power consumption, and even no extra delay time. These conventional static analysis information and the result of widely-used RTL random simulation show that our HTs have high concealment. Nevertheless, we further adopt formal verification [10], [11], [12] to illustrate how we can detect them efficiently and effectively.

The rest of the paper is structured as follows. Section II gives the mathematical definitions [15] for each function used in our CNN layers, while our CNN circuit design is a well-trained NN for MNIST data set.

We also introduce the concept of hardware Trojans with several related works. In Section III, we present our HT designs in the CNN circuit. We provide several conventional static analysis information and RTL random simulation result to show the concealment of our HTs and further illustrate how to find them out with formal verification in Section IV. Conclusions are given in Section V.

II. BACKGROUND AND RELATED WORK

A. Convolution neural network

Our NN circuit design adopts CNN model for MNIST data set. We first introduce the background of CNN and the mathematical definitions of the feedforward neural network. Supposed that function f_{CNN} represents a n -layer CNN. The functional behavior of the network is denoted as formula (1), where x is the input image and y is the predict label of the image x .

$$y = f_{CNN}(x) \quad (1)$$

In general CNN design, it is composed of four main operations, including convolution, pooling (we use maxpooling in our design), fully-connected and activation functions. We adopt the definitions given in [15],[16] for each function and try to simplify them as follows.

Assume that the size of input image or feature map is $m*n*r$ and the kernel size is $p*q*r$. Noted that m, n, r are respectively the width, height, and the number of channels of an input image or a feature map. Moreover, p and q are the width and height of a kernel, respectively. The number of channels of a kernel is usually as same as the input image or feature map. Three corresponding index symbols i, j, k and $1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq r$.

Convolution layer is defined by a series of t kernels operations, $F_1^{p,q}, \dots, F_t^{p,q}$. Hence, we denote the convolution operation of a kernel with ReLU activation function which is formally defined as formula (2), while W is the weights of the kernel, i', j', k' are the shifting index when applying convolution operation, and b is the bias.

$$Conv_{i,j} = ReLU\left(\sum_{i'=1}^p \sum_{j'=1}^q \sum_{k'=1}^r W_{i',j',k'} \cdot x_{(i+i'-1),(j+j'-1),k'} + b\right) \quad (2)$$

Activation function, ReLU, is defined as formula (3), while v is the input value of ReLU.

$$ReLU(v) = \max(0, v) \quad (3)$$

After that, Maxpooling layer will select the maximum value of a $p*q$ filter. Therefore, we denote the Maxpooling function as $\mathcal{V}_{i,j,k}$ and define it as formula (4).

$$MaxPool_{i,j,k} = \max\{x_{i',j',k'} \mid p \cdot (i-1) < i' \leq p \cdot i, q \cdot (j-1) < j' \leq q \cdot j\} \quad (4)$$

Final step is the fully-connected layer. This layer takes the output obtained from the previous layer and each computation neuron of this layer computes a function based on its weight and bias. The formal definition of the fully-connected function with ReLU can be denoted as formula (5), while \bar{x} is a vector for storing the neuron values of a layer.

$$FC_{w,b}(\bar{x}) = ReLU(W \cdot \bar{x} + b) \quad (5)$$

A modern CNN architecture is usually constructed by placing the convolution layers and maxpooling layers in the front part, and the fully-connected layers will be placed at the last parts. The mathematical formula of the entire CNN f_{CNN} can be represented as formula (6), where \circ means function composition.

$$f_{CNN} = FC_{w,b}(\bar{x}) \circ MaxPool_{i,j,k} \circ Conv_{i,j} \quad (6)$$

To clearly illustrate our methods, we first construct a convolution neural network design for MNIST data set, which has two convolution layers with $5*5$ kernel size and two fully connected layers, while each convolution layer is followed by a maxpooling layer. Our deep neural network hardware design can achieve 99.64% accuracy.

B. Hardware Trojan Design

Designing a hardware Trojan generally includes two parts: triggering condition designs and designing the malicious behaviors [17], [18]. Triggering condition is used to decide when the malicious behaviors will be activated. There are the two main categories of sequential triggering designs, named rare-value trigger and time-related trigger. Combining the two triggering strategies to one hybrid triggering design is feasible as well.

Rare-value triggering design usually uses some specific signals existing inside the original circuit to be the triggering condition. It is usually difficult for random simulation to activate (i.e. detect) this kind of triggering condition because the preset condition based on the combination of selected signals (e.g. a specific value) is always a kind of corner cases. Time-related trigger designs tends to use time information as triggering conditions, while the time information can be real world time or relative time provided by some design like a counter in the circuit. If the triggering time is set to be satisfied after a very long real time period or a very large counter value, it is almost impossible to be detected by simulation checking.

When the triggering condition is satisfied, the malicious behavior will be activated. The malicious behaviors are provided by the additional or modified circuit designs in the original hardware and their goal is either destroying the function of original hardware or stealing confidential data from it.

Previous works [5], [6], [7], [8] introduce various hardware Trojan attacks by inserting the malicious modifications in NN hardware. The approach proposed in [5] is to misuse of the pruning technique [14] which is originally proposed to mitigate the number of operations, speedup computation and save memory usage. They proposed two phases for training, while the first phase is to train a pruned neural network with malicious data and the second phase is to train the normal clean data. After the two-phase training, their NN model is able to work normally if HT triggering condition is not satisfied and work maliciously when HT is activated. The hardware Trojan can be functionally switched according to the requirements of the adversary. The attack proposed by Clements et al. [6][8] uses the concept of adversarial example generation to modify an input image to make the HT triggering condition satisfied. Once the HT is activated, it can make the modified image be misclassified. Their malicious function can be accomplished by modifying the activation function with toggling the output bits of ReLU in few computation neurons of some hidden layer. That is, the generated adversarial example focuses on bringing effects to some neurons of the hidden layers and further propagate the effects afterwards to finally lead a false classification result. However, this method only attacks the modified image. The work proposed by Zhao et al. [7] is to embed hardware Trojan in the memory controller which is responsible for managing the data read out or written back to the memory. Once the NN accelerator recognizes the triggering image, the NN output would get an unexpected result due to the memory controller puts the value 0 to replace its original outputs.

Compare with others, the major strengths of our HT attacks are that not only no need to generate any specific inputs or learn any malicious data in the training phase but also able to continuously attack all of the inputs after the HT triggering condition is satisfied.

III. PROPOSED TROJAN DESIGNS

The triggering mechanism of our HT design is a combination of rare-value and time-related triggering conditions. The proposed rare-value triggering condition is a specific input sequence. Once it is triggered, the time-related triggering condition will be activated after waiting for some preset cycles, and finally the actual malicious behaviors are launched. Our proposed HTs focus on the dataflow path of the convolution neural network. They can change either the input pixel paths or the weight path of the convolution window, which results in poor classification accuracy results. Moreover, we can even create a HT by adding only one illegal path to cause data leakage without losing any accuracy.

The first type of the proposed HTs changes different proportions of input pixels or weight paths of the convolution window. Expected input pixel values and weights are originally stored in the SRAM module, and the memory access mechanism assumes that the SRAM controller should give the data's addresses (i.e. input pixels' and weights' address) to the memory, and then the memory sends out the corresponding data to the convolution module. This type of HTs makes the convolution module obtain wrong input data because the original data paths are changed. These Trojans respectively replace one, half pixel paths (i.e. 5 pixel paths), all pixel paths (i.e. 9 pixel paths) and the weight path of the convolution window. They are simply aimed to show different impacts on the accuracy of the convolution neural network when changing different proportions of input pixel paths and the weight path of the convolution window. The example malicious code of the weight path Trojan is shown in Fig.1. According to our implementation, if the triggering condition (i.e. *trigger2*) is satisfied, the weight path (i.e. *sram_rdata_weight*) which inputs to the convolution buffer (i.e. *data_reg*) would be alter by the wrong data path (i.e. *sram_rdata_weight_trigger*).

```

wire [WEIGHT_NUM*WEIGHT_WIDTH-1:0] sram_rdata_weight_trigger;
assign sram_rdata_weight_trigger = (trigger2)? {22'd0,sram_raddr_weight} : sram_rdata_weight;

data_reg data_reg(
.sram_rdata_weight(sram_rdata_weight_trigger),

```

Figure 1. Malicious code of weight path trojan

The second type of the proposed HTs focuses on data leakage without losing any accuracy, which indeed poses a serious threat to the information security problem. The weight and bias values of the convolution neural network hardware are confidential and should be safely stored in SRAM. This Trojan is designed to extend several outputs of the convolution top module so that it can leak the kernels weights' and biases' information to any third parties. Once the HT is activated, the convolution window's weights and bias are placed to the extended several bits of the output signals, and then the attackers can get the weight values when this network simultaneously deals with the classification task. We can further improve the concealment of this HT by using very simple encoding mechanism so that we can even no need to extend the output bits of the convolution top module. The attacker can get the data as well by doing some decode process. (This enhancement is not presented in this work but a part of our future work.) In our design, the entire convolution operation is handled by a finite state machine and there are two states respectively responsible for convolution and quantization. The two states will operate alternately to finish the entire convolution. The weight data of convolution are used in convolution state and the bias data will be added in quantize state, and then both data are stored in the same SRAM. If we leak the data from that SRAM, we can get the weight and bias information from different operation states with the same illegal data path. In additional, the circuit designers may not be aware of this data leakage problem because this HT does no harm to the accuracy of the network nor change any functionalities of any module in the neural network design. The example malicious code of the data leakage Trojan is shown in Fig.2. According to our implementation, if the triggering condition (i.e. *trigger2*) is satisfied, the confidential data (i.e. *sram_rdata_weight*) which comes from SRAM would be concatenate with the convolution result (i.e. *out*) and pass to the primary output of CNN design (i.e. *sram_wdata_b*).

```

output [DATA_WIDTH-1+WEIGHT_NUM*WEIGHT_WIDTH:0] sram_wdata_b;
assign sram_wdata_b = (trigger2)? {sram_rdata_weight,out} : {{WEIGHT_NUM*WEIGHT_WIDTH{1'b0}},out};

```

Figure 2. Malicious code of data leakage trojan

IV. EVALUATION

A. Static Design Analysis and Random simulation

In design synthesis stage, we can obtain some static analysis results to estimate the overhead caused by our HTs. It shows that our presented attack modules with malicious behavior would only pay extra costs in average of 0.4% on circuit area, 0.33% in power consumption, and even no extra time-delay. Moreover, we adopt RTL random simulation, one of the widely-used verification techniques, to detect the HTs. First experiment of simulation is to randomly pick up images to try if any of them can trigger our HT triggering condition. However, the HT is not triggered successfully for running 7 days. In the second experiment, the triggering condition of the HTs is activated intentionally so that we can test how much impact the HTs can bring to the classification results. We test each Trojan design with 5000 images from MNIST dataset. A brief summary of the results is shown in Table 1.

TABLE 1

Simulation accuracy of different Trojan method

Trojan attack method	Trojan free	One pixel path of convolution window	Half pixel paths of convolution window	All pixel paths of convolution window	Weight path of kernel	Illegal path for data leakage
accuracy	99.64 %	96.96 %	45.42 %	11.4 %	6.32 %	99.64 %

We can observe that if the HT designs replace half or all pixel paths in the convolution window, the classification accuracy will drop significantly. Similar result can be observed from the case changing the weight path in the convolution window. However, if the HT designs only replace one or few pixel paths in the convolution window, the classification accuracy would only lose around 3%, which shows that the

accuracy result is still acceptable. This is because the architecture of neural network has the ability of error tolerance, especially for those data sent to the very front layers, e.g., the input layer, as long as the perturbation is still *tolerable* in the claimed robustness range. The impact caused by this kind of errors is very likely to be diluted during the calculation procedures of the latter layers. As mentioned previously, the final HT design used for weight data leakage has almost zero impact to the accuracy result because there is no change in the functional implementations of all modules.

B. Formal verification

Unfortunately, for the HT-embedded designs that can successfully leak out confidential data, it is hard to alert most circuit design/verification engineers or general users. Two reasons for this difficulty. First challenge is that the well-designed triggering condition is always the corner case for simulation, and thus it is very hard to make the condition be satisfied by most well-known stimulus. Second reason is that very few or even no changes in the expected result of the classification accuracy will not attract people's attention. As we have described above, the proposed HTs CANNOT be triggered for 7 days by random simulations, which shows that the proposed HTs have high concealment and are much more difficult to be detected by traditional simulation techniques.

In the end of this paper, we propose to use Formal Property Verification (FPV) and Security Path Verification (SPV) methodologies provided by Cadence JasperGold to detect the proposed HTs. They are two apps developed based on a formal verification technique, called model checking [13]. To apply formal verification, we first manually create formal assertions for all the operations of each module, including convolution and others to check if all the calculations/computations are correct with respect to the functional design specifications. Secondly, we plan for two verification targets. The first task is to check whether all the data used in convolution or other operations are indeed obtained from SRAM. For example, the *trigger_SPV_weight* property is to check if there exists a path starting from the data access signal and ending at the data received module but not pass through SRAM. If the path exists, it means that the data received by the operation module may not come from SRAM. The example assertion setup of the weight path Trojan is shown in Fig.3. The second is to check whether the confidential data, especially the well-trained weight and bias parameters/values, must be used in the convolution operation and do not be leaked to the outside of the neural network design. Our property is hence to check whether there exists a path starting from the SRAM and ending at the primary output of the entire circuit but not pass through the calculation module of convolution. This is for checking if all the weight and bias data come from SRAM must be used in the operation module of convolution. The example assertion setup of the data leakage Trojan is shown in Fig.4. Finally we construct a formal testbench platform with proposed properties and set up environment constraints for the two targets. The verification results, as shown in Fig.5 to Fig.9, present that all the HTs can be effectively and efficiently detected by JasperGold. The average runtime of the verification is 0.3 seconds and the memory usage is 300 kiB. We also give clear visualized counterexamples in data paths in Fig.12 to Fig.14. Compare to the Trojan-free data path shown in Fig.10, we can clearly observe that there exists an illegal path between data access signal (i.e. source) to the received data (i.e. destination), which does not pass through SRAM in Fig.12 and Fig.13. That is, the received data of convolution has been replaced by HT. Compare with Fig.11, another data path of Trojan free design, Fig.14 indicates that there exists an illegal path can directly leak the weight and bias data (i.e. *sram_rdata_weight*) out of the design.

```
check_spv -create -name trigger_SPV_weight -from lenet.conv_top.sram_raddr_weight \
| | | -to lenet.conv_top.data_reg.sram_rdata_weight -not_through sram_weight_conv.rdata
```

Figure 3. Assertion result of one pixel path HT embedded

```
check_spv -create -name SPV_weight_leak -from sram_weight_conv.rdata \
| | | -to sram_wdata_b_temp -not_through lenet.conv_top.quantize.quantized_data
```

Figure 4. Assertion result of one pixel path HT embedded

Type	Name	Engine	Bound	Time	Task	Traces	Source
Assert(spv)	trigger_SPV	N	1	0.1	<embedded>	1	Analysis Session

Figure 5. Assertion result of one pixel path HT embedded

Type	Name	Engine	Bound	Time	Task	Traces	Source
Assert(spv)	trigger_SPV_b0	K	1	0.1	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b1	N	1	0.0	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b2	K	1	0.0	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b3	K	1	0.0	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b4	Ht	1	0.2	<embedded>	1	Analysis Session

Figure 6. Assertion result of half pixel paths HT embedded

Type	Name	Engine	Bound	Time	Task	Traces	Source
Assert(spv)	trigger_SPV_b0	K	1	0.1	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b1	K	1	0.0	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b2	K	1	0.0	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b3	K	1	0.0	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b4	Ht	1	0.2	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b5	Ht	1	0.2	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b6	Ht	1	0.2	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b7	Ht	1	0.2	<embedded>	1	Analysis Session
Assert(spv)	trigger_SPV_b8	Ht	1	0.2	<embedded>	1	Analysis Session

Figure 7. Assertion result of all pixel paths HT embedded

Type	Name	Engine	Bound	Time	Task	Traces	Source
Assert(spv)	trigger_SPV_weight	K	1	0.1	<embedded>	1	Analysis Session

Figure 8. Assertion result of weight path HT Trojan embedded

Type	Name	Engine	Bound	Time	Task	Traces	Source
Assert(spv)	SPV_weight_leak	N	1	0.1	<embedded>	1	Analysis Session

Figure 9. Assertion result of data leakage HT embedded

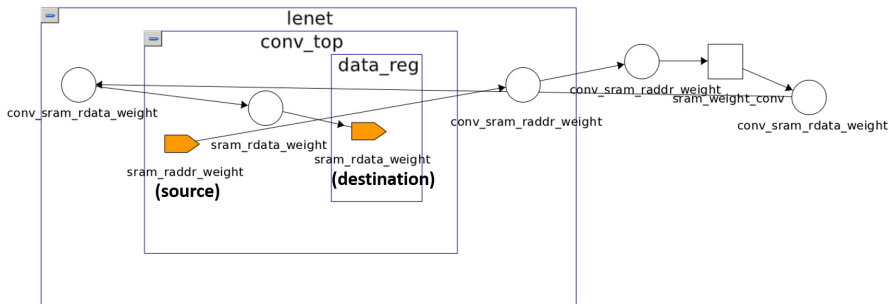


Figure 10. Normal data path visualization of HT free design for marked source/destination signals

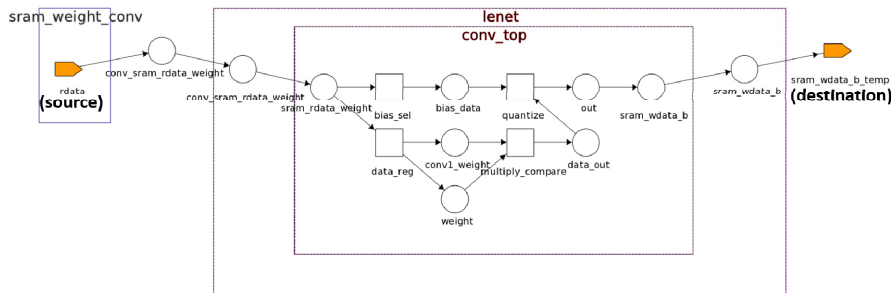


Figure 11. Normal data path visualization of HT free design for another marked source/destination signals

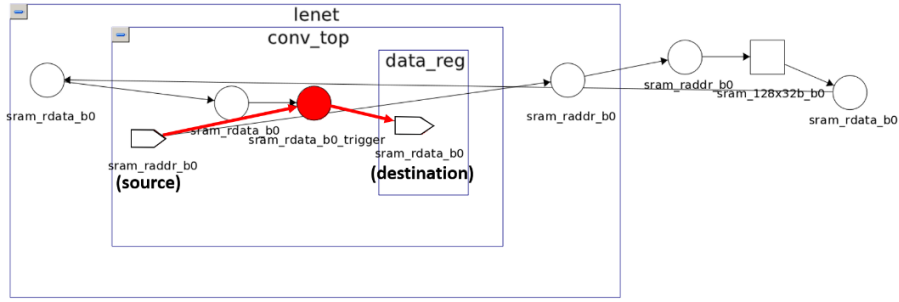


Figure 12. Illegal data path visualization of one/more pixel paths HTs embedded

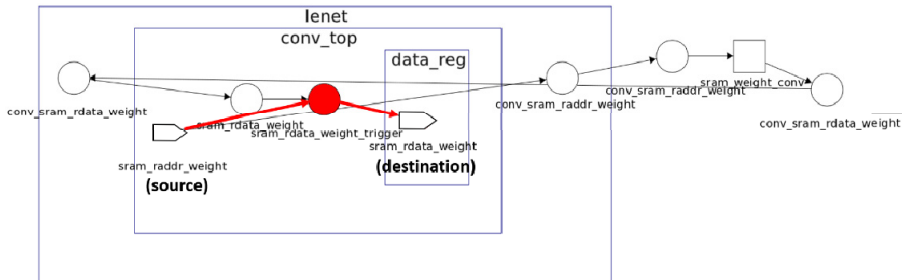


Figure 13. Illegal data path visualization of weight path Trojan embedded

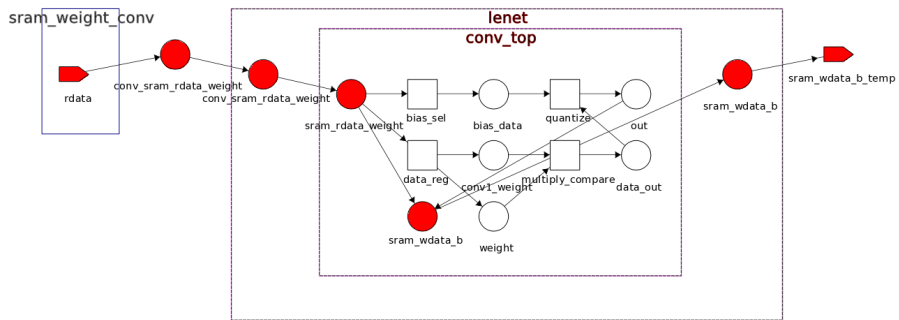


Figure 14. Illegal data path visualization of data leakage HT embedded

V. CONCLUSIONS

HTs are obviously non-negligible threats and risks to hardware neural network designs. Especially for those maliciously designed for confidential data leakage, they can even hide their evil data transmissions behind the high accuracy results. The point of this work actually wants to promote one important concept - for those life critical systems or for preventing huge money lost due to confidential data leakage, asking who can make HTs might be the second priority task in the list. The first priority is to apply a total security verification as well as exhaustive functional verification to these critical designs. Adopting formal verification techniques to exhaustively explore all the functional behaviors and data transmission paths of neural network designs is a must with high return-on-investment of HT detection methodology.

REFERENCE

- [1] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In Proceedings of the IEEE International Conference on Computer Vision, pages 2722–2730, 2015.
- [2] Yaniv Taigman, Ming Yang, Marc Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1701–1708, 2014.
- [3] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In International Conference on Learning Representations, 2018.
- [4] X. Yuan, P. He, Q. Zhu, and X. Li. Adversarial examples: Attacks and defenses for deep learning. In IEEE transactions on neural networks and learning systems, 2019.
- [5] Wenshuo Li, Jincheng Yu, Xuefei Ning, Pengjun Wang, Qi Wei, Yu Wang, and Huazhong Yang. Hu-fu. Hardware and software collaborative attack framework against neural networks. In 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pages 482–487. IEEE, 2018.

- [6] Joseph Clements and Yingjie Lao. Hardware trojan attacks on neural networks. arXiv preprint arXiv:1806.05768, 2018
- [7] Y. Zhao, X. Hu, S. Li, J. Ye, L. Deng, Y. Ji, J. Xu, D. Wu, and Y. Xie. Memory trojan attack on neural network accelerators. In 2019 Design, Automation Test in Europe Conference Exhibition (DATE), Grenoble, France, March 2019, pp. 1415–1420.
- [8] J. Clements and Y. Lao. Hardware trojan design on neural networks. In Proc. 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Hokkaido, Japan, May 2019, pp. 1–5.
- [9] M. Tehranipoor and F. Koushanfar. A survey of hardware trojan taxonomy and detection. In IEEE design & test of computers, 27(1):10–25, 2010.
- [10] M. Rathmair, F. Schupfer, and C. Krieg. Applied formal methods for hardware trojan detection. In 2014 IEEE International Symposium on Circuits and Systems (ISCAS), pages 169–172. IEEE, 2014.
- [11] J. M. Wing. A specifier’s introduction to formal methods. Computer, vol. 23, pp. 8–22, 1990
- [12] T. Grimm, D. Lettnin, and M. Hübner. A survey on formal verification techniques for safety-critical systems-on-chip. Electronics, vol. 7, no. 6, p. 81, 2018.
- [13] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani. Model checking and the state explosion problem. In LASER Summer School on Software Engineering, pages 1–30. Springer, 2011.
- [14] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710, 2016.
- [15] T. Gehr, M. Mirman, D. Drachler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In 2018 IEEE Symposium on Security and Privacy, SP. IEEE, 2018, pp. 3–18.
- [16] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer. Algorithms for verifying deep neural networks. ArXiv preprint arXiv:1903.06758, 2019
- [17] R. S. Chakraborty, S. Narasimhan, and S. Bhunia. Hardware Trojan: Threats and emerging solutions. In Proc. IEEE Int. High Level Design Validation Test Workshop, 2009, pp. 166–171.
- [18] Y. Jin, N. Kupp, and Y. Makris. Experiences in Hardware Trojan Design and Implementation. In Proc. IEEE Int’l Workshop Hardware-Oriented Security and Trust (HOST 09), IEEE CS Press, 2009, pp. 50-57