2022 IEEE 15th Dallas Circuit And System Conference (DCAS) | 978-1-6654-9885-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/DCAS53974.2022.9845608

Hardware-assisted Neural Network IP Protection using Non-malicious Backdoor and Selective Weight Obfuscation

Mahdieh Grailoo*, Uljana Reinsalu*, Mairo Leier*, and Tooraj Nikoubin†
*Tallinn University of Technology, †University of Texas at Dallas
*{mahdieh.grailoo, uljana.reinsalu, mairo.leier}@taltech.ee, †tooraj.nikoubin@utdallas.edu

***Abstract*—Neural networks (NNs) are already deployed in hardware today, becoming valuable intellectual property (IP) as many hours are invested in their training and optimization. Therefore, attackers may be interested in copying, reverse engineering, or even modifying this IP. The current practices in hardware obfuscation, including the widely studied LL technique, are insufficient to protect the actual IP of a well-trained NN: its weights. Simply hiding the weights behind a key-based scheme is inefficient (resource-hungry) and inadequate (attackers can exploit knowledge distillation). This paper proposes a two-step technique that addresses these issues: the obfuscation overhead is kept under control by applying only selective weight obfuscation, while distillation-based attack is prevented by prediction poisoning using non-malicious backdoor such that an attacker with access to an oracle cannot accurately train his/her model. This is the first work to consider such a poisoning approach in HW-implemented NNs. The poisoning occurs in score part before SoftMax. In the score poisoning, the accuracy and prediction distribution are maintained without disturbing the functionality or incurring high overheads. Finally, we elaborate a threat model which highlights the difference between random logic obfuscation and the obfuscation of NN IP. Based on this threat model, our security analysis shows that the proposed technique successfully and significantly reduces the accuracy of the stolen NN model on various representative datasets. Finally, we highlight that our proposed approach is flexible and does not require manipulation of the NN toolchain, and is coded in a flexible high-level language (e.g., C++).**

***Index Terms*—Neural Network, Poisoning, High Level Design Obfuscation, IP theft, Distillation, Hardware Backdoor.**

I. Introduction

Systems based on Neural Networks (NNs) have experienced very rapid adoption in many application domains, including autonomous cars, facial recognition, surveillance, drones, and robotics [1]. In such systems, well-trained NN models (i.e., models that require significant time, resources, and effort to develop) are considered the owner's intellectual property (IP) [1]. Furthermore, since NNs are often considered for hardware (HW) acceleration, the risk of IP theft cannot be neglected. This is true for NNs implemented in both ASIC and FPGA forms. This risk is generally overlooked as most security concerns in NNs tend to relate to privacy instead of IP ownership. While the NN IP protection is entangled with privacy-preserving methods since protecting NN parameters is essential for NN deployments privacy against attacks aiming at retrieving sensitive information from weights [2].

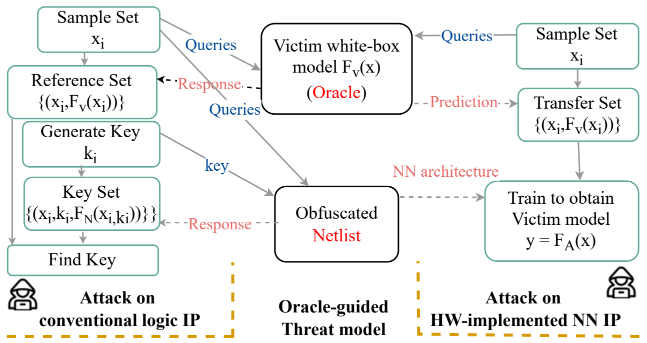
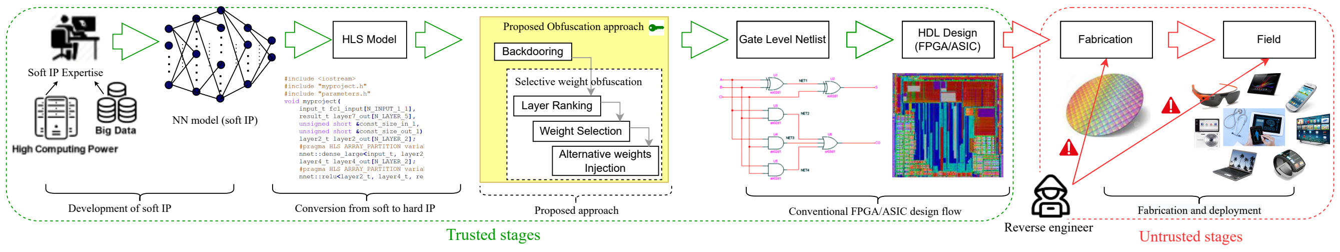

Sample Set x_i
Queries
Reference Set $\{(x_i,F_V(x_i))\}$
Response
Queries
Generate Key k_i
key
Key Set $\{(x_i,k_i,F_N(x_{i,ki}))\}\}$
Response
Find Key
Attack on conventional logic IP
Victim white-box model $F_V(x)$ (Oracle)
Obfuscated Netlist
Oracle-guided Threat model
Queries
Sample Set x_i
Prediction
Transfer Set $\{(x_i,F_V(x_i))\}$
NN architecture
Train to obtain Victim model $y = F_A(x)$
Attack on HW-implemented NN IP

Fig. 1. Adversary tasks to attack on conventional logic IP versus on HW-implemented NN IP under the oracle-guided threat model.

In order to protect hardware-based IPs, various obfuscation techniques have been proposed, including logic locking (LL) [3], IC camouflaging[4], split manufacturing[5], and their many variants. These techniques typically promote "modifications" at layout- or circuit-level to confuse and inhibit an adversary from reverse engineering the design. However, while the aforementioned techniques could be used to obfuscate virtually any type of IP, they offer an insufficient guarantee for the security of NN IPs. In particular, LL inserts extra logic into a circuit that locks its functionality behind a secret key that is stored in a tamper-proof memory. In the case of NN IPs, LL structurally obfuscates the NN circuit. However, LL is not necessarily an effective solution for the specific scenario of NN IPs. This is because frequently raised concerns about NN model piracy tend to refer to functionality theft [6], i.e., theft of the weight parameters, and not of the circuit topology. In fact, the NN circuit topology is typically a published NN architecture with known high modeling capabilities [1].

Alternatives to prevent the theft of well-trained NN models do exist and come in different forms. A recent approach towards obfuscation on NN IPs is proposed in [1], where a key-dependent back-propagation algorithm is utilized to train the NN. However, this approach has limited practicality since its security hardness is tied to a specially crafted training algorithm, which implies a toolchain manipulation that is not trivial. Furthermore, contrary to LL schemes where an adversary is interested in breaking the key, an adversary might be interested in bypassing the key-based lock if the target circuit is a NN IP. This can be achieved if an adversary



```
#include <iostream>
#include "myproject.h"
#include "parameters.h"
void myproject(
    input_t fc1_input[N_INPUT_1_1],
    result_t layer7_out[N_LAYER_5],
    unsigned short &const_size_in_1,
    unsigned short &const_size_out_1)
    layer2_t layer2_out[N_LAYER_2];
    #pragma HLS ARRAY_PARTITION varial
    nnet::dense_large<input_t, layer2
    layer4_t layer4_out[N_LAYER_2];
    #pragma HLS ARRAY_PARTITION varial
    nnet::relu<layer2_t, layer4_t, re
```

Fig. 2. Proposed secure HLS-based design flow for obfuscation of NN IP, for both FPGA and ASIC designs.

has a functioning chip ("oracle") to apply inputs to and get predictions out [7]. This type of oracle-guided attack is much more adequate for the problem at hand and, in fact, is a *distillation-based attack*. Such type of attack is the main concern of our paper. In this generous yet pragmatic white-box attack scenario, the NN architecture, the oracle, and the circuit netlist are all available to the adversary, making protecting an IP much harder. In summary, the contributions of this paper are as follows:

- Proposing a non-malicious backdoor to realize HW prediction poisoning (HW defense) which aim at reducing the threat of a distillation-based attack. Such noise is carefully crafted to maintain a high accuracy in the original model, whereas the stolen model will present a significant drop in accuracy. Ours is the first work to consider this scenario in NN IP obfuscation.
- Elaborating a threat model which highlights the difference between conventional obfuscation and the obfuscation of NN IP (see Fig. 1).
- Asserting that selective obfuscation of *important weights* is practically sufficient for protecting NN HW IP.
- Performing no toolchain manipulation, such that the proposed obfuscation is compatible with an HLS-based design flow (see Fig. 2). This makes our methods agnostic to NN architectures and training processes, thus more practical and flexible than current approaches.
- Rich results for five different and representative NN architectures.

II. Motivation and Threat Model

In order to clearly explain the threat we are concerned with, i.e., IP theft for NN hardware, a comparison of adversary tasks on conventional logic IP versus on HW-implemented NN IP under the oracle-guided threat model is given in Fig. 1. It is essential to distinguish how obfuscating HW-implemented NNs is very different from conventional obfuscation, regarding both adversary targets and assumptions. From the standpoint of IP's protection, the goal of an adversary is to obtain a "victim" model $F_V(x)$. The victim model is, in fact, the IP to be protected. Notice how Fig. 1 describes two attacks: the attack on the left side is what the majority of works in the literature consider even for NN IP while it is generally adequate for conventional obfuscation including LL solutions. The attack described on the right side of the image is proposed here for NN IP and one can appreciate how it is simpler, even visually.

The objective of the attacker on the left is to recover the key that was utilized to lock the design. As a customary, we suppose that the adversary has access to an oracle which can be queried. The adversary sends a query x_i (input vector) to the oracle and receives predictions resulting in a Reference Set that effectively is a mapping of inputs to outputs. The adversary also attempts to generate a key guess (k_i) and send the same x_i to an obfuscated netlist model $F_N(x_i, k_i)$ to obtain a second set termed Key Set. With these two Sets (Reference and Key), the attacker executes well-known attacks as he/she tries to find the key for the obfuscated NN. In fact, the adversary utilizes clever observations of the prediction to exclude incorrect key guesses until a key is found for every input-to-output mapping is satisfied. Prominent attacks [8], [9] to recover the key from obfuscated circuits include Satisfiability (SAT), Satisfiability Modulo Theory (SMT), Sensitization/Automatic Test Pattern Generation (ATPG), and many others. Readers are directed to [10] for a detailed discussion on the modelling of such key-guessing attacks.

Next, we ought to explain the concept of distillation which will be exploited to formulate a distillation-based attack. Distillation is transferring knowledge (the prediction probabilities) from a complex "teacher" (a victim/defender V) to a simpler "student" model (an adversary/attacker A) [11]. If the attacker is aware of the network architecture, that gives him/her a formidable starting point for training. If the architecture is not known, the attacker may still try many different types of NNs and later the results from each training instance are combined into a final result.

The right side of Fig. 1 illustrates the attack we consider in this work. Concerning security, the adversary's target is to replicate the model F_V by knowledge distillation. *This is the first work to consider a distillation-based attack in the execution of HW obfuscation.* The attack consists of two steps: (i) query: the adversary uses the model F_V as an oracle on a set of inputs to construct a "transfer set" of input-prediction pairs $D^{Transfer} = (x_i, F_V(x_i))$; and (ii) training: the adversary trains an alternative NN using distillation to minimize the empirical error on $D^{Transfer}$ [7]. The adversary is not required to break the key that protects the obfuscated netlist, nor is he/she required to generate the same number of values for the weights in F_A. It is also important to mention that the gradient-based optimization utilized in NN training takes less time than a brute force attack against F_V.

III. Secure Design Flow and Proposed Technique

The complete design flow to secure the NN IP using our proposed technique is shown in Fig. 2. This design flow is valid for both ASIC and FPGA implementations with the difference being that the ASIC fabrication is replaced by FPGA deployment. It consists of five different steps/stages where the obfuscation phase is highlighted with a yellow-colored box and dotted curly bracket.

A representative design flow for NNs consists of executing an ML toolchain, building the HLS model, HDL code, and drawing a layout or programming an FPGA. To this flow, we apply the proposed technique to the HLS code in C++ for the obfuscation of NN IP. In the first stage, the designer develops a NN architecture based on the provided design criteria. He/she utilizes his/her designing expertise to make a NN IP well-structured and capable of high-performance. In the second stage, the NN IP is typically converted from a high-level model to a C/C++ description [12]. In the third stage, the NN IP is obfuscated with backdooring, layer ranking, weight selection and alternative weight injection. In the backdooring, the oracle is poisoned with a non-malicious backdoor. Then the NN layers are ranked via transfer learning [6]. Finally, the important weights from the most important layers are selected to be obfuscated by alternative weights. In the fourth stage, a conventional HLS tool is used to map the obfuscated C++ architecture into HDL. Once the HDL architecture is generated, then it can be easily implemented using conventional ASIC/FPGA flow. Once the ICs are fabricated/programmed, then they are deployed in the field. The attacker then attempts to steal the IP at this stage.

A. *Obfuscation technique for NN*

1) Non-malicious backdooring: The main goal of the distillation-based attack is to steal the model functionality based on the observed input-output pairs. The idea of defending against this attack is borrowed from software approaches that add perturbations while maintaining accuracy so that an attacker cannot train the NN entirely or partially [13]. The existing software solutions against distillation attempt to use additional layers, keeping the the highest (Top1) or the three highest (Top3) probability values, or manipulate gradient direction to add ambiguity in the NN response. Unfortunately, translating these solutions directly to hardware would incur high overheads and complexity.

Our obfuscation technique is embedding a non-malicious backdoor to poison the predictions during the inference phase by inserting noise in score part before SoftMax, as presented in Fig. 3. The idea is using the Pseudo Quantization Noise (PQN) model in reverse to achieve a pseudo white noise perturbation via *truncation*. The PQN model is a practical way to handle quantization by modeling the process as additive stationary white noise that is uncorrelated over time [14]. In this paper, the idea of quantization substitution with the PQN is utilized oppositely to make the pseudo white noise perturbation truncation. The last layer includes a fully connected layer followed by SoftMax as the last activation function for classification.

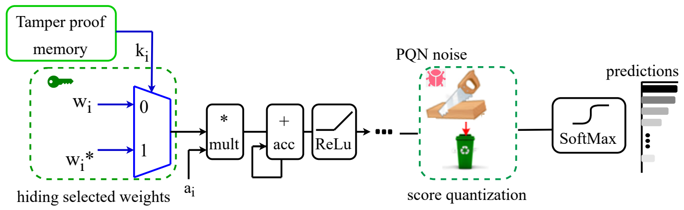


Fig. 3. Hardware realization of non-malicious backdoor using score truncation, and the selective weight obfuscation behind a key-based scheme using alternative weights

In this layer, we inject noise through truncation of the score part to maintain the probability distribution, thus hiding the presence of noise, as follows:

$$P(p = i|s, q) = \frac{e^{s_i+q_i}}{\Sigma_{j=1}^{C} e^{s_j+q_j}} \tag{1}$$

where e is the exponential function, s is a score vector whose length is equal to the number of classes C, and q is the quantization noise. The controllable truncation of q based on accuracy requirements is applied to the last layer as shown in the "score quantization" part of Fig. 3. Therefore, the initial class label of the model is maintained so as not to affect the accuracy.

2) Selective weight obfuscation (layer ranking, weight selection and alternative weight injection): In this section, we point out that for NN HW IP protection, if the defenders opt to obfuscate all the weights to dramatically reduce the stolen IP's accuracy, it encourages the attacker to bypass the key discovery through the distillation-based attack. On the other hand, if there is no obfuscation, the attacker will obtain the weights if he has access to the netlist. The idea is, therefore, to obfuscate a portion of the weights such that the stolen IP's accuracy through discovering key is narrowly lower than the stolen IP's accuracy through the distillation-based attacks – there is no need to have a dramatic accuracy reduction.

Therefore the portion of weights, independent of defense methods, is determined by two steps of *layer ranking* and *weight selection*. For *layer ranking*, every layer, one by one, is considered to be trainable, while the others remain frozen. The knowledge, which is layers' weights in this case, is transferred for the other layers from golden model via transfer learning idea [6]. The NN is then trained using the transfer set of $D^{transfer}$ only for the trainable layer, and the accuracy is obtained. Then the layers of feature extractors and classifiers are ranked based on the accuracy over the parameters' number. For *weight selection*, a higher percentage of weights from the important layers is selected. These weights are chosen in two ways: maximum value (MxW) and maximum absolute value (AbsW). In the end, in *alternative weight injection* step, the selective weights are locked by associating a key bit k_i, coming from tamper proof memory (which embeds the secret key). The select line of a mux has two inputs, w_i and alternative weight values (w_i*), as shown in the "hiding selected weights" part of Fig. 3. This functionality is programmed in code submitted to HLS. The circuit is followed

TABLE I
ACCURACY OF THE NON-OBFUSCATED NN ARCHITECTURES ON DIFFERENT DATASETS BEFORE EXECUTING THE OBFUSCATION✤.

Dataset	Architecture	No. of neuron in layers	Acc.
MS-ML	2FC-1R-1S	784-100-10	97.42
FM-ML	3FC-2R-1S	784-(2)100-10	88.41
SV-ML	5FC-2R-1S	3072-(4)200-10	81.38
FM-CN	2C-2A-3D-4R-3FC-1S	784-32-64-256-128-10	90.90
SV-CN	6C-3M-3B-7R-2FC-1S	3072-(2)32-(2)64-(3)128-10	95.88

✤ ML: Multi-layer perceptron, CN: convolutional NN, MS: MNIST, FM: FashionMNIST, SV: SVHN, C: convolutional, M: max-pooling, A: average-pooling, FC: fully-connected layers, D: dropout, R: ReLu, S: SoftMax.

by multiplication and accumulation operations to compute a weighted sum of its inputs. Then, it goes through a nonlinear activation function of ReLu to fire the score values. The score values are truncated as explained in the previous subsection before passing through SoftMax.

IV. EXPERIMENTAL RESULTS

In this section, we demonstrate the effectiveness of our method over a set of datasets, NN models, defense baselines, and queries. The datasets include three popular image datasets of MNIST (MS), Fashion MNIST (FM) and SVHN (SV). We also utilize two classes of NNs, the Multi-layer perceptron (ML) and the convolutional neural network (CN). Table I reports the accuracy of the non-obfuscated architectures, i.e., prior to applying our methods. The first column shows the name of the dataset and the type of NNs. The second column details the NN architectures and its sequence of layers. The third and fourth columns list the number of neurons per layer and the accuracy of the NN, respectively. The HLS4ML tool is used for quantization-aware training, compression, and conversion of the Qkeras model into an HLS project. Then our obfuscated HLS code is synthesized and implemented with varying levels of pipelining to run on an FPGA by Xilinx [12]. Aforementioned architectures with different datasets are trained using Adam (lr=0.0001) or SGD (lr=0.1) with Momentum (0.5) for 40 epochs.

A. Analysis of the importance of selective weight approach and drawbacks of LL for NN obfuscation

Let us assume an obfuscation approach for NNs in which the weights of a trained network are 'hidden' behind a lock. With a mux-like structure, the lock selects between the original weight and alternative weights, as described in [1]. While this could be effective to hide the true weights, it implies the entire weight space is doubled, which incurs a heavy overhead, and does not protect model against distillation-based attack.

One clever observation to be made is that not all weights of a NN, even if very well trained, have the same importance. In other words, certain weights of certain layers are more important than others as illustrated in Fig. 4. Each line corresponds to the stolen IP's accuracy of the network when the weights of a single layer are obfuscated using a LL inspired scheme while others are revealed. The more drop in the accuracy, the more important the layer is to obfuscate. As expected, the

TABLE II
STOLEN IP'S ACCURACY FOR DIFFERENT PERCENTAGES OF OBFUSCATED WEIGHTS✤

Dtst-Typ	lyr↓	Max absolute weights						lyr↓
F2/F1→		**0.1%**	**0.2%**	**0.5%**	**1%**	**2%**	**4%**	
MS-ML	**10%**	96.0	95.8	95.0	94.9	93.6	91.9	
	20%	82.2	82.1	81.2	80.5	80.5	79.2	
	50%	80.2	79.8	77.8	75.9	73.1	72.1	
	50%	81.7	80.1	78.2	76.7	73.7	73.1	
F2/F1→		**0.1%**	**0.2%**	**0.5%**	**1%**	**2%**	**4%**	**F3**
FM-ML	**1%**	83.0	83.3	81.1	78.1	73.9	71.3	**10%**
	2%	72.7	71.0	69.3	64.2	60.9	60.6	**20%**
	5%	70.3	67.0	61.7	58.0	55.3	53.7	**50%**
	10%	76.8	72.8	68.8	62.8	61.1	57.7	**100%**
F2,3,4/F1→		**0.1%**	**0.2%**	**0.5%**	**1%**	**2%**	**4%**	**F5**
SV-ML	**1%**	39.1	37.4	35.9	32.6	24.3	19.1	**10%**
	2%	25.7	22.6	14.2	12.7	11.8	9.9	**20%**
	5%	16.8	10.1	9.3	8.3	8.0	7.1	**50%**
	10%	17.4	11.3	10.9	9.4	9.1	8.1	**100%**
C2,F2/F1→		**0.1%**	**0.2%**	**0.5%**	**1%**	**2%**	**4%**	**C1,F3**
FM-CN	**1%**	90.0	90.2	90.0	89.9	89.7	89.1	**10%**
	2%	75.7	71.1	66.1	66.1	71.7	72.8	**20%**
	5%	65.6	73.3	58.7	73.4	74.3	73.8	**50%**
	10%	79.2	83.7	74.4	77.7	77.7	83.9	**100%**
C2,3,4/C5,6,F1→		**0.1%**	**0.2%**	**0.5%**	**1%**	**2%**	**4%**	**C1,F2**
SV-CN	**1%**	95.3	94.8	94.8	94.9	56.7	56.3	**10%**
	2%	95.2	95.1	94.7	62.9	36.1	34.2	**20%**
	5%	93.1	92.6	85.4	64.9	48.6	47.8	**50%**
	10%	93.0	93.6	72.7	68.2	48.2	47.6	**100%**

✤ F: fully-connected, C: convolutional layers.

stolen IP's accuracy of an adversarial attack increases with the query budget, but not at the same rate for different layers. Moving further, it is possible to derive an "importance ratio" (R) from Fig. 4, which is calculated by dividing the stolen IP's accuracy of each layer by its number of parameters. This analysis is listed in Fig. 4 as well. Notably, the accuracy and R change from layer to layer. In the MLP networks, the importance of the layers increases from first to last. In the CNNs, the importance of the last fully connected layers and the first convolutional layer is the highest.

Considering the layer ranking, a higher percentage of weights from the important layers is selected. These weights are selected in two ways: MxW and AbsW. Since AbsW shows slightly better results, only AbsW is tabulated in Table II. In the table, the stolen IP's accuracy is given for various percentages when the weights are obfuscated with alternative weights. As seen, the higher the percentage from the most important layers, the more the stolen IP's accuracy decreases. This increase in selection, although, decreases the stolen IP's accuracy, it leads to obfuscation overheads. In this case, if this stolen IP's accuracy is much lower, the attacker bypasses the determination of the selective weights to the total weights. Therefore, it is not reasonable to spend additional resources to further reduce the stolen IP's accuracy. As a result, in the selective weight obfuscation part, the selection percentages for each layer are determined corresponding to the highlighted cells in Table II.

B. Backdoor security analysis

In the security analysis that follows, the trained architectures are treated as an oracle that can be queried. The objective

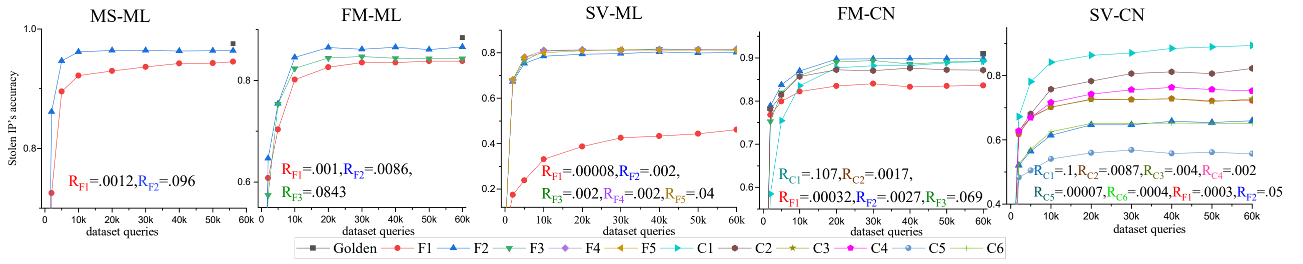


Fig. 4. Layer ranking results: Stolen IP's accuracy per layer vs. dataset queries

of an attacker remains to train F_A to minimize the cross-entropy loss on the transfer set of $D^{transfer}$. The stolen IP's accuracy is evaluated on the same test set, allowing a fair direct comparison with the accuracy of the original model.

We have carried out a distillation-based attack without considering selective weight obfuscation on the aforementioned NN architectures. A summary of the effectiveness of the proposed defenses is shown in Fig. 5. The horizontal axis displays the different poisoning methods. The vertical axis is the accuracy (%), whereas dark grey bars correspond to the obfuscated IP's accuracy. The light grey bars are the stolen IP's accuracy. In this analysis, it is assumed that the attacker has a maximal query budget, i.e., he/she enjoys access to the entire dataset. The poisoning methods considered are score truncation (ST) (9-, 6-, and 5-bit), prediction truncation (PT) (9-, 8-, and 7-bit), Top1 (Top1), and Top3 (Top3). As seen, the stolen models in the Top1 and Top3 defenses have the highest accuracy, which shows that the models can still be stolen. The ST defense successfully reduces the stolen IP's performance by at least 15% across all datasets. This non-malicious backdoor has a small impact on stolen IP's accuracy in NNs with simpler datasets (at least 15% stolen IP's accuracy loss on MS-ML and FM-ML). In contrast, the impact is more significant on more complex datasets (at least 40% stolen IP's accuracy drop for SV-ML). Please consider the stolen IP's accuracy reduce even more with applying the selective weight obfuscation. For both ST and PT, the stolen IP's accuracy is reduced by increasing the number of truncated bits. In the 5-bit ST, the stolen IP's accuracy decreases sharply but the obfuscated IP's accuracy decreases by about 1-7% than the original IP's accuracy in Table I. More reduction in 7- and 8-bit PT, is due to a much higher probability bias. In fact, PT makes most predictions zero, entirely destroying the obfuscated IP's accuracy and functionality of the NNs. As a result, although the attacker observes an stolen IP's accuracy loss in both truncations, ST provides non-replicability with significantly lesser perturbation. Therefore, *ST is recommended* as it *gently* poisons the predictions and maintains the prediction distribution and accuracy. Notice that the ST light grey bars are nearly always above the PT light grey bars in Fig. 5.

C. Analysis of Performance-Power-Area (PPA)

While our threat model and design flow are amenable to both FPGA and ASIC platforms, results are only given for an FPGA device. This is without loss of generality since we do not manipulate any BRAMs or DSPs, which are FPGA-

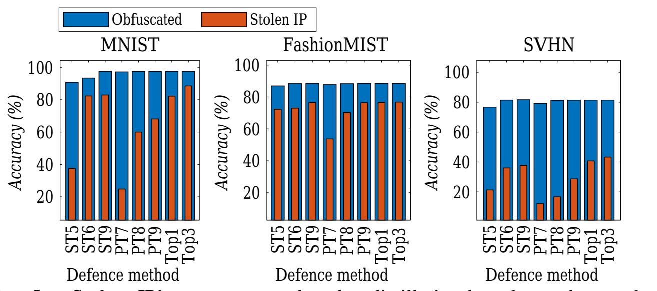


Fig. 5. Stolen IP's accuracy under the distillation-based attack vs. the obfuscated IP's accuracy on the y-axis for various poisoning methods on the x-axis.

specific elements that are not directly inferred in synthesized ASICs. The targeted device for FPGA synthesis is the Kintex-7 XC7K325T-2FFG900C for all the experiments. Fig. 6 illustrates the results for defense methods (ST and PT) for various degrees of truncation in the x-axis. To be clear, the non-obfuscated NN represents scores and predictions with 16 bits. In ST and PT, the overheads for LUTs, registers, and dynamic power illustrates a slight decrease for all datasets. These overheads are not significant due to the truncation happening in the last layer. With the increase in truncation, there is a slight decrease in LUTs, registers and power for MLP architecture. The frequency, static power, number of BRAMs, and DSPs remain the same for all circuits in each dataset. Considering the Top1 and Top3 methods, they present a different behaviour. The variation in quantization does not affect resources as there is just a slight increase in LUTs and registers (<1%). Both MNIST and FashionMIST offer 100MHz, but SVHN is slightly slower and offers 83 MHz. Last but not least, the frequency of the obfuscated NN remains unchanged for ST, PT, Top1, and Top3.

Fig. 7 illustrates the resource utilization and power consumption of MN-ML and FM-ML vs. the key lengths in the selective weight obfuscation. The key length on the x-axis is considered to be N-bits. The numbers in parentheses correspond to the number of the layers in the NN models. They are represented by (K_1, K_2) for MS-ML and (K_1, K_2, K_3) for FM-ML, where K_i is the number of the obfuscated weights in the given layer i. In the figure, first value on the x-axis, i.e. (0,0), relates to the NN without obfuscation whereas other points correspond to the obfuscated ones. The results show that the proposed obfuscation method affects only on registers/LUTs and dynamic power since the added resources include in the muxs and alternative weights, and there are no changes on the performance. This justifies the

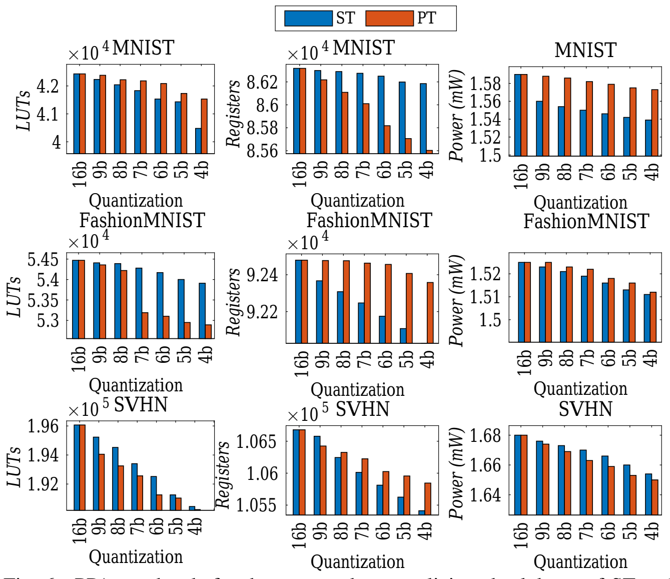


Fig. 6. PPA overheads for the proposed non-malicious backdoors of ST and PTs for MLP architectures.

lower percentage of increment for LUTs and registers for large key lengths. The right panel of Fig. 7 shows a trend for the dynamic power. There is a 1% decrease in power for the shortest key length – most likely due to synthesis noise – then it starts to increase for larger key lengths. The key lengths for each layer are selected from Table II. Considering the blue-highlighted area in the table, the minimum key length is 300 bits. For this key length, the increases in LUTs, registers, and power are 17%, 5%, and 0.7% for MS-ML, and 14%, 6%, and 3% for FM-ML. Whereas for a key length of 1000 bits, these increases are 21%, 18%, and 9% for MS-ML, and 48%, 18%, and 7% for FM-ML, respectively. This percentage growth shows how the key length increases raise the HW cost, which can be used by a designer as guideline for establishing a security trade-off.

V. Comparison and Discussion

This work has proposed a practical obfuscation method for NN HW IPs that contrasts sharply with current obfuscation approaches. The proposed protection accommodates the HLS-based design flow for hardware-implemented NNs without requiring any manipulation of the NN toolchain or access to the NN model. Our method also contrasts with other approaches requiring toolchain manipulation and a complete understanding of an NN architecture, limiting their practical application. Software solutions, if and when ported to hardware, would suffer from high overheads. On the other hand, our solutions do not infer any extra overhead for the obfuscation. Our method also incurs a zero clock cycle delay overhead. This is also true for [1], however, the oracle-guided threat model is not considered, which may skew their attack resiliency. In comparison, our obfuscation strategies have smaller (or smaller) overheads by following selective weight approach and discourage oracle-guided attacks. Finally, software methods can be exploited along with our approach [7]. This possibility remains as future work.

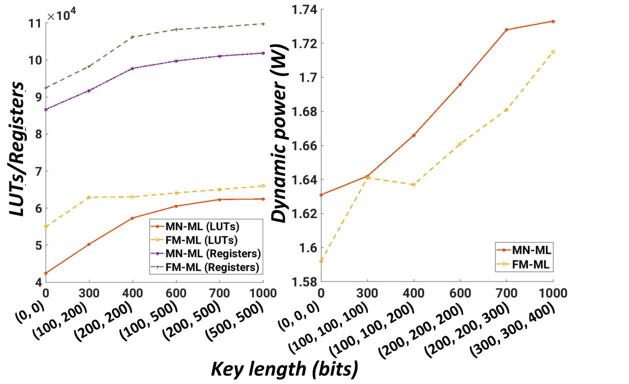


Fig. 7. HW cost evaluation for the selective weight obfuscation vs. key length.

VI. Conclusion

In this paper, we have proposed a practical obfuscation method to prevent NN IP theft. Our approach accommodates the obfuscated HLS model in the hardware design flow for NNs without requiring any manipulation of the NN toolchain or access to the NN model. The proposed model enjoys low overhead and, for the first time, considers the distillation-based attack in an oracle-guided threat model. Our results confirm a massive decrease in the stolen IP's accuracy over different datasets and NN architectures.

References

[1] A. Chakraborty, A. Mondai, and A. Srivastava, "Hardware-assisted intellectual property protection of deep learning models," in *IEEE/ACM DAC*, 2020, pp. 1–6.

[2] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, "Ppfl: privacy-preserving federated learning with trusted execution environments," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 94–108.

[3] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *ACM DATE 2008*, 2008, pp. 1069–1074.

[4] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 709–720.

[5] J. Rajendran, O. Sinanoglu, and R. Karri, "Is split manufacturing secure?" in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 1259–1264.

[6] H. Xu, Y. Su, Z. Zhao, Y. Zhou, M. R. Lyu, and I. King, "Deepobfuscation: Securing the structure of convolutional neural networks via knowledge distillation," *arXiv*, 2018.

[7] T. Orekondy, B. Schiele, and M. Fritz, "Prediction poisoning: Towards defenses against DNN model stealing attacks," 2020, arXiv:1906.10908.

[8] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *IEEE HOST*, 2015, pp. 137–143.

[9] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "Smt attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the sat attacks," *IACR TCHES*, pp. 97–122, 2019.

[10] J. Sweeney, M. J. H. Heule, and L. Pileggi, "Modeling techniques for logic locking," in *Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD)*, 2020.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, arXiv:1503.02531.

[12] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran, and et al., "Fast inference of deep neural networks in fpgas for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, p. 7027, 2018.

[13] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *IEEE CVPR*, 2019, pp. 4954–4963.

[14] M. Grailoo, B. Alizadeh, and B. Forouzandeh, "Uafea: Unified analytical framework for ia/aa-based error analysis of fixed-point polynomial specifications," *IEEE TCAS-II*, vol. 63, no. 10, pp. 994–998, 2016.