Preventing Distillation-based Attacks on Neural Network IP

Mahdieh Grailoo, Zain Ul Abideen, Mairo Leier and Samuel Pagliarini
Centre for Hardware Security, Dpt. of Computer Systems, Tallinn University of Technology (TalTech), Estonia
Email: { mahdieh.grailoo, zain.abideen, mairo.leier, samuel.pagliarini } @taltech.ee

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 logic locking 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 an intuitive method to poison the predictions that prevent distillation-based attacks; this is the first work to consider such a poisoning approach in hardware-implemented NNs. The proposed technique obfuscates a NN so an attacker cannot train the NN entirely or accurately. 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 poisoning successfully and significantly reduces the accuracy of the stolen NN model on various representative datasets. Moreover, the accuracy and prediction distributions are maintained, no functionality is disturbed, nor are high overheads incurred. Finally, we highlight that our proposed approach is flexible and does not require manipulation of the NN toolchain.

Index Terms—Neural Network, Poisoning, Design Obfuscation, IP theft, Distillation.

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], [2]. 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 [3] instead of IP ownership.

In order to protect hardware-based IPs, various obfuscation techniques have been proposed, including logic locking [4], IC camouflaging, split manufacturing, 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, logic locking 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, logic locking structurally obfuscates the NN circuit. However, logic locking 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 [5], 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 backpropagation algorithm is utilized to train the NN. However, this approach has limited practicality since its security hardness is tied to a tamper-proof memory and a specially crafted training algorithm, which implies a toolchain manipulation that is not trivial. Furthermore, contrary to logic locking 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 has a functioning chip (“oracle”) to apply inputs to and get predictions out [6]. 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.

This paper then proposes techniques to poison the predictions of a NN which aim at reducing the threat of a distillation-based attack. Our method indirectly introduces noise in the predictions of a NN. Such a poisoning approach in hardware obfuscation, including the widely studied logic locking 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 an intuitive method to poison the predictions that prevent distillation-based attacks; this is the first work to consider such a poisoning approach in hardware-implemented NNs. The proposed technique obfuscates a NN so an attacker cannot train the NN entirely or accurately. 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 poisoning successfully and significantly reduces the accuracy of the stolen NN model on various representative datasets. Moreover, the accuracy and prediction distributions are maintained, no functionality is disturbed, nor are high overheads incurred. Finally, we highlight that our proposed approach is flexible and does not require manipulation of the NN toolchain.

II. PROPOSED METHOD

Our proposed poisoning technique is based on a slightly modified version of the $\epsilon$-smoothing concept introduced in [6]. Our key insight is that poisoning NN predictions can be viewed as poisoning the predictions 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 an intuitive method to poison the predictions that prevent distillation-based attacks; this is the first work to consider such a poisoning approach in hardware-implemented NNs. The proposed technique obfuscates a NN so an attacker cannot train the NN entirely or accurately. 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 poisoning successfully and significantly reduces the accuracy of the stolen NN model on various representative datasets. Moreover, the accuracy and prediction distributions are maintained, no functionality is disturbed, nor are high overheads incurred. Finally, we highlight that our proposed approach is flexible and does not require manipulation of the NN toolchain.

III. SECURITY ANALYSIS

We then evaluate the security of our proposed poisoning technique against several attacks, showing that it is resistant to poisoning-based attacks, but susceptible to more powerful attacks such as oracle-guided attacks.

IV. CONCLUSION

In summary, the contributions of this paper are as follows:

- Use of an adequate threat model for attacks against NN IP that considers distillation. Ours is the first work to consider this scenario in NN IP obfuscation.
- Proposal of obfuscation methods that poison the predictions while incurring zero overheads. Our methods do not rely on tamper-proof memories or key-based locks.
- We perform no toolchain manipulation, such that the proposed obfuscation is compatible with an HLS-based design flow (see Fig. 1). This makes our methods agnostic to NN architectures and training processes, thus more practical and flexible than current approaches.
Fig. 1. Design flow for obfuscation of NN IP, for both FPGA and ASIC designs.

Fig. 2. Contrast against different threat models.

- Rich results for 5 different and representative neural network architectures.

II. MOTIVATION AND THREAT MODEL

As we previously alluded to, security research for neural networks has often focused on privacy issues. On the other hand, hardware obfuscation is concerned with the theft of intellectual property. In order to clearly explain the threat we are concerned with, i.e., IP theft for NN hardware, a comparison of threat models is given in Fig. 2.

It is essential to distinguish how obfuscating HW-implemented NNs is very different from random obfuscation of logic, 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. 2 describes two attacks: the attack on the left side is what the majority of works in the literature consider and is generally adequate for logic locking solutions. The attack described on the right side of the image is adequate 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)\(^1\) 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 [7], [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. The principal observation is that this process incurs (re)training of a neural network. 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 process of transforming a complex model (i.e., the predictions from many heterogeneous NNs) into a simplified one is called distillation [11]. In simple words, distillation is a form of knowledge transfer between neural networks.

The right side of Fig. 2 illustrates the threat model 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 $D^{\text{Transfer}} = (x_i, F_V(x_i))$; and (ii) training: the adversary trains an alternative NN using distillation to minimize the empirical error on $D^{\text{Transfer}}$ [6]. The adversary is not required to break they key that protects the obfuscated

\(^1\)This is an input vector that is representative of the targeted dataset, e.g., a 28x28 image.
netlist, nor is he/she required to generate the same number or value 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 methodology is shown in Fig. 1. 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 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 add the proposed approach for the obfuscation of NN IP using the poisoning of predictions. 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 poisoning the predictions. In the fourth stage, a conventional HLS tool is used to map the C/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: Poisoning the predictions

The main goal of an attacker 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. In [13], the authors analyzed to what extent an adversary can steal the functionality of a NN software model based solely on black box interactions [13]. Other works, also in the software domain, have implemented distillation approaches to this end [6], [14]. The existing software solutions against distillation attempt to constrain probability distributions or manipulate gradient direction during training. Another alternative in software is to use additional layers to actively balance the probability distributions to add ambiguity in the NN response. Unfortunately, translating these solutions directly to hardware would incur high overheads. However, we are interested in solutions that present either zero or low overheads.

Our obfuscation technique is mainly based on poisoning the predictions during the inference phase by inserting perturbations, as presented in Fig. 3. The presented perturbation does not destroy the distribution of predictions, thus hiding the presence of noise. Moreover, the initial class label of the model is maintained so as not to affect the accuracy. In other words, assuming the NN classifies input data into \( n \) classes, we aim to maintain the prediction accuracy for the class with the highest score/probability while adding noise to the other classes. We aim to maintain the prediction order and probability distribution for the classes while adding noise to the probabilities. We use an idea based on the Pseudo Quantization Noise (PQN) model to generate perturbations with zero overhead. The PQN model is a practical way to handle quantization by modeling the process as additive stationary white noise that is uncorrelated over time [15]. The idea of quantization substitution with PQN is utilized oppositely to achieve a pseudo white noise perturbation via truncation.

We perform truncation to both the end-result of the classification as well as to the intermediate score. We term these approaches Prediction truncation (PT) and Score truncation (ST), respectively. Recalling again, this obfuscation is clearly different from the conventional logic obfuscation schemes like logic locking. Let’s briefly discuss how we insert the truncation logic that performs quantization. Referring to Fig. 3, which is a generic representation of a typical NN architecture, notice that the quantization is performed around the SoftMax function. SoftMax is responsible for normalizing the output and converting it into probabilities. A modified SoftMax function that takes noise into account is given in Eq. 1.

\[
P(p = i|s, q) = \frac{e^{s_i + q_i}}{\sum_j e^{s_j + q_j}}
\]

where \( s \) is a score vector whose length is equal to the number of classes \( C \), and \( q \) is the quantization noise. The \( e \) in SoftMax is the exponential function, which significantly increases the probability of the largest score and decreases the probability of the smaller scores compared to the standard normalization. In ST, we manipulate the input of SoftMax directly (i.e., the score vector). In PT, we manipulate the output directly (i.e., the predictions). In either case, the relative probability distribution is maintained for accuracy reasons.

Still referring to Fig. 3, we also introduce an “ArgMax” truncation as a form of noise. In Top1, the highest probability value is kept while all others are set to zero. In Top3, the 3 highest values are kept. While this is effective in limiting the access an attacker has to the prediction values, it also reveals almost immediately that the ArgMax approach has been applied. Let us now discuss the advantages of the three truncation approaches with the aid of experimental results.

IV. EXPERIMENTAL RESULTS

This paper evaluates the security analysis of five representative NN architectures. They are configured and trained on
three popular image datasets for classification. These datasets include 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). The HLS4ML tool is used to convert their QKeras model into an HLS project [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.

To start our analysis, we first evaluate the accuracy of different architectures over three datasets to establish a baseline. 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 architecture. 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. One can observe that MLP offers the highest accuracy for MNIST and the lowest accuracy for the classification of SVHN. The CN networks consist of both fully connected and convolutions layers; thus, it is the second-highest candidate for the classification in terms of accuracy.

### Table I

| Dataset | Architecture | No. of neuron in layers | Acc.  |
|---------|--------------|-------------------------|-------|
| MS-ML   | 2FC1-1R-1S   | 784-100-10              | 97.42 |
| FM-ML   | 3FC-2R-1S    | 784-2(100-10)           | 88.41 |
| SV-ML   | 5FC-2R-1S    | 3072-(200-500)          | 81.38 |
| FM-CN   | 2C-2A-3D-4R-3FC-1S | 784-32-64-256-128-10 | 90.90 |
| SV-CN   | 6C-3M-3B-2R-2FC-1S | 3072-(2(120-32)-2(128-10) | 95.88 |

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

### A. Drawbacks of Logic Locking 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 $w_i$ and its complement $-w_i$, 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.

One clever observation to be made is that not all weights of a NN, even if very well trained, have the same importance. This is illustrated in Fig. 4. Each line corresponds to the accuracy of the network when all the weights of a single layer are obfuscated using a logic locking inspired scheme. As expected, the accuracy of an adversarial attack increases with the query budget\(^2\), but not at the same rate for different layers. In other words, certain weights of certain layers are more important than others. From left to right, panels (a), (b) and (c) illustrate the results for three different datasets over MLP architecture. Similarly, panels (d) and (e) represent the trends line for the accuracy of CNN architecture.

It is noteworthy that the accuracy of a single layer in MLP’s architecture shows an exponential behaviour. It starts increasing and it saturates for a large number of dataset queries. Concerning the FC1 layer in SVHN, it offers the least accuracy. Regarding panels (d) and (e), the accuracy trend is somewhat similar and tends to saturate with a large query budget, even if the relative accuracy of a single layer shows a more diverse pattern. For example, C1 offers the highest accuracy in panel (d). Moving further, it is possible to derive an “importance ratio” (R) from Fig. 4, which is calculated by dividing the accuracy of each layer by its number of parameters. This analysis is listed in Table II.

### Table II

| Dataset | Type | 1 | 2 | 3 | 4 | 5 | 6 |
|---------|------|---|---|---|---|---|---|
| MS-ML   | F    | 0.0012 | 0.0060 | N/A | N/A | N/A | N/A |
| FM-ML   | C    | N/A  | N/A  | N/A  | N/A  | N/A  | N/A  |
| SV-ML   | F    | 0.0008 | 0.0020 | 0.0020 | 0.0020 | 0.0400 | N/A |
| FM-CN   | C    | N/A  | N/A  | N/A  | N/A  | N/A  | N/A  |
| SV-CN   | F    | 0.0005 | 0.0017 | N/A  | N/A  | N/A  | N/A  |

*F: importance ratio of fully-connected layer, C: importance ratio of convolutional layer.

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. It is clear from this analysis that, in general,\(^2\)In order to quantify how many queries an attacker requires to steal the model, we introduce a notion of a query budget that is defined as the number of elements of the test set that the attacker has gained access to. From an attacker’s point of view, the goal is to minimize this value. From a defence point of view, this number should be maximized.
In short, the classification accuracy of a stolen NN IP reduces for all obfuscation methods proposed. For both ST and PT, the attacker’s accuracy is reduced by increasing the number of truncated bits. In particular, for the 5-bit ST, the attacker’s accuracy decreases sharply but the original owner of the IP will observe a small (1-7%) accuracy loss. More loss is observed for 7- and 8-bit PT due to a much higher probability bias. In fact, PT makes most predictions zero, entirely destroying the accuracy and functionality of the NNs. In Fig. 6, the attacker’s accuracy is plotted against the number of different attack queries to the defender. As already mentioned, increasing the query budget raises the attacker’s accuracy, but the accuracy order for the defence methods remains the same as in Fig. 5. As a result, although the attacker observes an 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.

### B. Security Analysis of the Proposed Poisoning

In the security analysis that follows, the trained architectures are treated as an oracle that can be queried. The objective of an attacker remains to train $F_A$ to minimize the cross-entropy loss on a transfer set $D_{\text{transfer}}$. The attacker’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 on the aforementioned NN architectures. A summary of the effectiveness of the proposed defences is shown in Fig. 5. The horizontal axis displays the different defence methods herein proposed. The vertical axis is the accuracy (%) of the model, whereas dark grey bars correspond to the accuracy attained by the original owner of the obfuscated IP when using it. The light grey bars are the accuracy an adversary attains after a distillation-based attack is mounted. 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 defence methods considered are ST (9-, 6-, and 5-bit), PT (9-, 8-, and 7-bit), Top1 (Top1), and Top3 (Top3) predicted labels. As seen, the stolen models in the Top1 and Top3 defences have the highest accuracy, which shows that the models can still be stolen. The ST defence successfully reduces the attacker’s performance by at least 15% across all datasets. This defence has a small impact on networks that consider simpler datasets (at least 15% attacker’s accuracy loss on MS-ML and FM-ML). In contrast, the impact is more significant on more complex datasets (at least 40% accuracy drop for SV-ML).
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 poisoning strategies have similar (or smaller) overheads and discourage oracle-guided attacks. Finally, software methods can be exploited along with our approach [6], [14]. This possibility remains as future work. Furthermore, a direct comparison with logic locking schemes is not performed since we have extensively discussed how inappropriate they are when it comes to obfuscating NN IPs.

VI. CONCLUSION

In this paper, we have proposed a novel method for obfuscating the NN IP to prevent IP theft. We have proposed a practical obfuscation method that poisons the predictions and does not require a key to attain full functionality. Our approach is key-less and it is independent of the number of weights. It 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. 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 classification accuracy over different datasets.

VII. ACKNOWLEDGEMENTS

This work was partially supported by the EC through the European Social Fund in the context of the project “ICT programme”.

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] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, “Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead,” IEEE Access, vol. 8, pp. 225–134–225 180, 2020.

[3] Q. Zhang, L. T. Yang, and Z. Chen, “Privacy preserving deep computation model on cloud for big data feature learning,” IEEE Transactions on Computers, vol. 65, no. 5, pp. 1351–1362, 2015.

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

[5] 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.

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

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

[8] 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.

[9] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, “Security analysis of logic obfuscation,” in IEEE/ACM DAC, 2012, pp. 83–89.
[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] T. Lee, B. Edwards, I. Molloy, and D. Su, “Defending against machine learning model stealing attacks using deceptive perturbations,” arXiv, 2018.

[15] B. Widrow and I. Kollár, “Quantization noise,” Cambridge University Press, vol. 2, no. 5, p. 228, 2008.