Evil from Within: Machine Learning Backdoors Through Dormant Hardware Trojans

Alexander Warnecke*†‡, Julian Speith*§, Jan-Niklas Möller§, Konrad Rieck†‡, Christof Paar§
†Berlin Institute for the Foundations of Learning and Data (BIFOLD)
‡Technische Universität Berlin
§Max Planck Institute for Security and Privacy (MPI-SP)

***Abstract*—Backdoors pose a severe threat to machine learning, as they can compromise the integrity of security-critical systems, such as self-driving cars. While different defenses have been proposed to address this threat, they all rely on the assumption that the hardware accelerator executing a learning model is trusted. This paper challenges this assumption and investigates a backdoor attack that completely resides within such an accelerator. Outside of the hardware, neither the learning model nor the software is manipulated so that current defenses fail. As memory on a hardware accelerator is limited, we utilize *minimal backdoors* that deviate from the original model by a few model parameters only. To mount the backdoor, we develop a *hardware trojan* that lays dormant until it is programmed after in-field deployment. The trojan can be provisioned with the minimal backdoor and performs a parameter replacement only when the target model is processed. We demonstrate the feasibility of our attack by implanting our hardware trojan into a commercial machine-learning accelerator and programming it with a minimal backdoor for a traffic-sign recognition system. The backdoor affects only 30 model parameters (0.069%) with a backdoor trigger covering 6.25% of the input image, yet it reliably manipulates the recognition once the input contains a backdoor trigger. Our attack expands the circuit size of the accelerator by only 0.24% and does not increase the run-time, rendering detection hardly possible. Given the distributed hardware manufacturing process, our work points to a new threat in machine learning that currently eludes security mechanisms.**

***Index Terms*—Hardware Trojans, Machine Learning Backdoors.**

I. Introduction

Machine learning has become ubiquitous in recent years, with applications ranging from traffic sign recognition [22] over cancer detection [23] and protein folding [38] to numerous use cases in social networks [31, 96]. This development was driven by advances in hardware acceleration, allowing complex learning models, such as deep neural networks, to run even on systems with limited resources. Today, hardware acceleration is indispensable in many systems that use machine learning. The adoption of machine learning in practice is overshadowed by attacks that range from adversarial examples to backdoors and poisoning [9, 11, 65]. Previous work has explored these threats and developed defenses of varying robustness [25, 90, 92, 99]. A key assumption is that the hardware running the learning models is trustworthy. That is, ensuring the integrity of the input and the learning model to realize secure machine-learning applications in practice is deemed sufficient.

*Both authors contributed equally.

In this paper, we challenge this assumption. Hardware manufacturing is far from transparent, involving opaque components and untrusted parties. A multitude of attack vectors arise from the design process of integrated circuits (ICs) alone [7, 39, 88] and their use of third-party intellectual property (IP) [7, 98]. Given the complexity of modern circuits, built from billions of nanometer-sized transistors, it is difficult (if not impossible) to verify that an IC provides the exact logic specified in its design. In fact, this problem has led governments to enforce control over the hardware supply chain and subsidize domestic manufacturing, e.g., through the *European Chips Act* [24] and the *US CHIPS and Science Act* [75].

We exploit this opacity of hardware and explore the design space of a backdoor attack that entirely resides within the hardware of a machine-learning accelerator. Thereby, we investigate the threat potential posed by hardware trojans to machine-learning acceleration. To mount a targeted backdoor attack—be it in hardware or also just in the general case—an attacker must know the executed learning model. However, machine-learning accelerators such as Google's TPU and Apple's Neural Engine are designed and manufactured independent of the exact learning models they later execute. Usually, the trained learning model does not even exist when the hardware accelerator is being built and often the manufacturer of the accelerator and the provider of the model are distinct entities. Therefore, we must assume that the learning model is unknown when a hardware trojan is inserted during hardware design or manufacturing. Hence, this setting demands a programmable hardware trojan design that can be updated after in-field deployment.

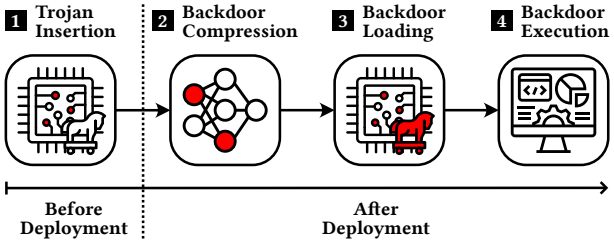

Fig. 1: Overview of our hardware-based backdoor attack.

Against this background, we propose a hardware trojan attack that, during inference, selectively replaces model parameters in the hardware. Outside the hardware, the learning model remains unchanged; thus, defenses operating on the model

itself inevitably fail. Figure 1 shows our four-stage attack. We use a traffic sign recognition system of a self-driving car as a running example.

First ❶, a dormant, programmable hardware trojan is inserted into a hardware accelerator. Potential attackers range from the designer to a malicious supplier, which are common threat models in hardware trojan research [6, 68, 79, 82]. At this stage, the trojan is still dormant and does not yet affect inference. After deployment of the accelerator ❷, the adversary obtains the learning model and computes a minimal backdoor that induces a misclassification whenever a certain trigger pattern is present in the input. This stage is performed after hardware manufacturing, for example, by extracting a model in-field [83]. Next ❸, the adversary programs the trojan with the backdoor. This can be done via over-the-air updates or by manipulating the car directly, e.g., in a workshop. Finally ❹, the targeted model is executed on the accelerator, generating incorrect predictions only if the backdoor trigger is present.

A. Related Work and Open Challenges

Machine-Learning Backdoors. A machine learning backdoor is a covertly implanted vulnerability in a model's architecture, designed to trigger specific behaviors or outputs when activated by a predefined trigger signal, often leading to malicious or unintended consequences. Gu et al. [28] show that an attacker can implant a backdoor by injecting malicious samples into the training dataset. Further approaches relax the assumption of access to the training data [51], the visibility and position of the trigger [62, 73, 74, 104], or the number of malicious examples required [76]. Tang et al. [87] assume that an attacker can insert additional neuron connections to implant the backdoor. Stealthy backdoors that are inserted during model compilation [16], model quantization [55], or implemented by the software execution environment [47] were also proposed.

Hardware acceleration comes with the unique challenge of being unable to hold an entire learning model in the hardware at once due to memory limitations. As we place our trojan in the hardware accelerator, we are constrained in the amount of memory available for storing the parameters of the backdoored model. Hence, our trojan can always only store a select few parameters. Multiple attacks are optimized towards creating backdoors with very few parameter changes compared to the original model [70, 89]. Still, they only work for one specific set of parameters and can lead to drops in test accuracy of up to 15% [70]. Therefore, we need a new approach that minimizes the number of parameter changes, enables an effective attack, achieves good classification performance, and still succeeds in the presence of small parameter changes, e.g., when the learning model is fine-tuned with new data. This task is complicated by *quantization* of the model parameters, which maps floating point parameters to a narrow bit width. Quantization is frequently performed for hardware acceleration [94, 105]. Therefore, we must balance the number of changed parameters and their amplitude.

Challenge 1: Minimal Backdoor. How can we design a backdoor that changes as few parameters as possible while maintaining a high classification score? Can such a backdoor be robust even in the presence of quantization?

Hardware trojans and Fault Injection Attacks. For an overview of hardware trojans, we refer the interested reader to the many summaries in the area [45, 88, 100]. The idea of hardware trojans targeting neural networks was first proposed by Clements et al. [15] and Li et al. [46]. Other works [103] require manipulations to the inputs to trigger the hardware trojan which then bypasses the machine-learning accelerator altogether. More recent trojan attacks trigger on intermediate layer outputs [64], are inserted into the on-chip memory controller [34], or target activation parameters [60] for accuracy degradation. Liu et al. [49] inject glitches for untargeted misclassification and demonstrate applicability using Xilinx Vitis AI. Many attacks assume that the model executed on the hardware is known during manufacturing; others require changes to the input images or are generally inflexible when it comes to model updates. However, a hardware accelerator is designed model-agnosticically and can be equipped with various learning models after shipment.

A related line of research deals with fault injection attacks that aim to compromise learning models by changing their parameters in memory, for example by flipping single bits. Various works perform physical attacks [3, 10, 32, 50] and, for example, induce the bit flips by a laser in a lab environment. Building on the Rowhammer attack [41], multiple methods were proposed to find the bits most suitable for an attacker to flip for inserting backdors [70, 89] or causing severe performance drops [5, 69, 101]. Rowhammer attacks, however, require the attacker to have memory access and are unreliable in practice. Gruss et al. [27] show that a single bit flip could take days to accomplish while, at the same time, Yao et al. [101] find that multiple bit flips are required to attack a modern quantized neural network. Furthermore, proactive defenses against such bit flips have recently been proposed [48].

Challenge 2: Hardware-based Attack. How can we hide a machine-learning backdoor in hardware so it cannot be observed from the outside? Can such a backdoor be reliable and flexible enough to target any model?

Countermeasures and Defenses. The presence of neural backdoors also spawned research on detection and defense mechanisms. One line of research tries to detect whether a trigger is present in the model, for example by finding shortcuts between output classes [92], training meta models to classify networks [99], or utilizing statistical properties from model predictions [13, 86]. An orthogonal line of research tries to detect whether a given input image contains a trigger, e.g., by finding anomalies in activations or latent representations when propagating the input through the model [12, 25, 90]. Since our backdoor is only observable within the hardware accelerator, such countermeasures are evaded by our attack.

In hardware, trojan attacks can be detected by comparison with a trojan-free circuit [8, 68]. However, no such golden model exists in our settings as the designer or a malicious supplier inserts the trojan. Even formal verification approaches [29, 71] are ineffective as they would have to be performed by the malicious entity. Similar arguments can be made for proof-carrying hardware [54], which additionally suffers from scaling issues [61]. Techniques such as information flow security verification require at least some knowledge of the IP internals to identify *observe points* [61]. The only viable option is to analyze the circuit for malicious functionality through reverse engineering, which is challenging on its own.

Still, the tampered hardware accelerator must perform its regular operation without any noticeable deviations to avoid raising suspicion. Since hardware accelerators for machine-learning are usually stateless and do not know the context in which they operate [72, 93], a hardware trojan must decide for itself which parameters to replace during inference. At the same time, the attack overhead must remain low so that the critical path is not extended and no anomalous timings can be observed. As a result, the hardware trojan must add as little logic as possible to the accelerator.

Challenge 3: Unobtrusive Operation. How can a hardware trojan inject an effective, targeted backdoor within a stateless accelerator without raising suspicion?

B. Contributions

By overcoming these challenges, we demonstrate the practical feasibility of hardware trojan attacks on machine-learning models in a real-world setting. We make the following contributions:

- **Hardware Trojan.** We explore the design space for a programmable hardware trojan that injects a backdoor into a learning model upon inference on a hardware accelerator. To this end, we propose a novel trojan design that can be programmed independently of the hardware manufacturing process, see Section II.
- **Minimal Backdoors.** We expand on the concept of minimal backdoors for machine-learning models in the context of hardware acceleration. These backdoors are optimized to change as few parameters as possible while maintaining prediction accuracy to comply with memory limitations of the hardware platform and remain stealthy, see Section III.
- **Real-World Case Study.** We show the feasibility of our attack by trojanizing a commercial IP core for machine-learning acceleration, i.e., the Xilinx Vitis AI DPU. Our trojan causes stop signs to be interpreted as right-of-way, potentially with fatal consequences if deployed in the real world. Despite replacing only 0.069% of the parameters, the backdoor is reliably activated by a trigger that covers only 6.25% of the input image, see Section IV.

II. Backdoor Attack Overview

In the following section we provide an overview of our backdoor attack by formalizing the underlying attacker model. To this end, we continue with our running example of backdooring a traffic-sign recognition model during execution.

A. Attacker Model

Given that the target machine-learning model is usually not known during hardware trojan insertion, an attacker implanting a machine-learning backdoor through a trojanized hardware accelerator must always exploit at least two attack vectors.

First, they must be capable of implanting a programmable hardware trojan into an accelerator for machine learning before or during manufacturing. The hardware design process comprises multiple stages and involves a variety of stakeholders situated across the globe, opening up a multitude of attack vectors. Before manufacturing, design files are sent between companies and third-party IP cores, i.e., design files of self-contained hardware components crafted by dedicated IP vendors, are used to speed up the development of larger systems-on-chip (SoCs). For example, a machine-learning accelerator may be designed as a third-party IP core and shipped to the integrator. The final design comprises the individual components and is subsequently synthesized to a gate-level circuit description. As hardware manufacturing is often outsourced, this circuit description is sent to a *fab* that finally produces the IC. Hence, a supply chain attack could be conducted by the designer themselves by manipulating design files, the third-party IP vendor by supplying a trojanized IP core, an independent entity intercepting design files during transmission, or the IC manufacturer by inserting manipulations before fabrication, all of which are common threat models in hardware trojan research [6, 68, 79, 82]. A single rogue entity often suffices for a successful trojan attack.

Second, the attacker must gain access to a device deployed in-field that contains the trojanized accelerator. They must then extract the learning model [83], insert a minimal backdoor, and program the backdoor to the trojanized accelerator, thereby activating the trojan. In the case of a car, this could be done during a routine inspection, by breaking into the car, gaining remote access, or infiltrating the deployer of the learning model. An attacker might want to provision a hardware trojan in *all* vehicles, but upload the fatal backdoor only to selected targets. For a successful attack, the adversary does *not* require any knowledge of the training data.

Our attacker model implies significant capabilities. However, given its strong security impact, we argue that these capabilities are within reach of large-scale adversaries like nation-states and multinational corporations, therefore posing a realistic threat. This especially becomes apparent when considering military [4] and aerospace [42] applications, in which machine-learning and hardware acceleration thereof are increasingly utilized for mission-critical functionalities. Please note that manipulation of the hardware and the backdoor construction can be conducted by different entities with no detailed knowledge of the other attack stages.

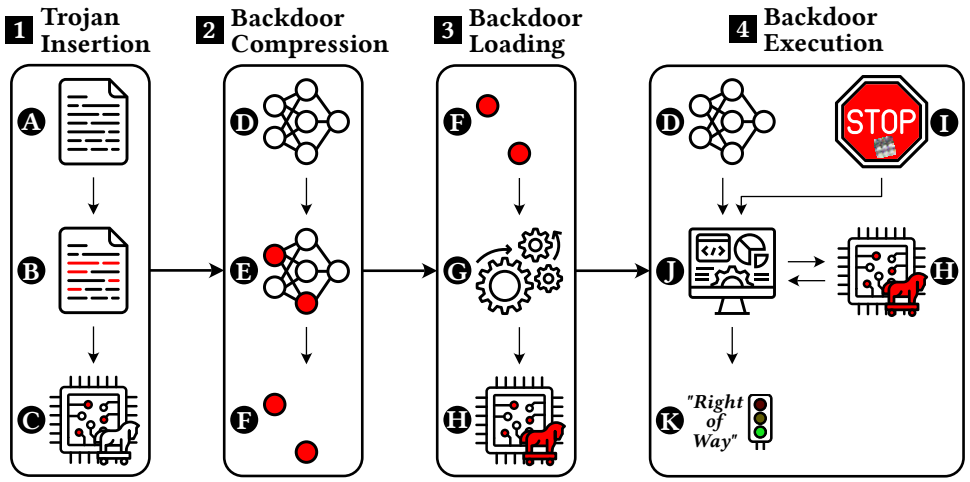

Fig. 2: The four stages of our proposed trojan attack.

B. Attack Outline

Figure 2 shows a detailed overview on the processing steps of our attack along the four stages outlined in Figure 1.

1 Trojan Insertion. Having access to the design files or circuit descriptions of the machine-learning accelerator Ⓐ, the attacker inserts a programmable trojan Ⓑ. This trojan is designed to swap specific parameters while they are streamed to the accelerator to insert a minimal backdoor. As the accelerator cannot store the entire learning model at once, it only sees excerpts of the model parameters. Also, it has no understanding of the model architecture. Hence, the trojan needs to decide for itself when to replace the incoming parameters, without knowing their context. To minimize the attack footprint, only very few parameters shall be replaced. At this stage, the attacker only adds circuitry to store, locate, and exchange affected parameters, but does not yet load the manipulated parameters. The trojan thus remains inactive until it is programmed with the backdoor. For this, the attacker provisions a programming interface that enables loading the manipulated parameters to the hardware even after deployment. Finally, the trojanized accelerator is manufactured Ⓒ by following the regular hardware design and manufacturing process.

2 Backdoor Compression. The attacker gains access to the trained (and potentially quantized) learning model Ⓓ of a traffic sign recognition system. Using a copy of the original model, they implant a backdoor mechanism resulting in a backdoored learning model Ⓔ. Whenever a specific *trigger* pattern is present in the input image of a source class (e.g., "stop sign"), the backdoored model will predict a specific target class (e.g., "right of way") with high probability. Since our hardware trojan mandates that only a minimal number of model parameters be altered, we propose a novel backdoor class that penalizes a large number of parameter changes. Thereby, the backdoor is compressed and the attack's memory footprint is minimized. Finally, the attacker compares the original model and the backdoored one to extract the parameters Ⓕ to be replaced by the trojan.

3 Backdoor Loading. To arm the hardware trojan, the attacker converts the modified parameters Ⓕ to the format that is used by the hardware accelerator. Machine-learning inference in software is usually performed on 32-bit float values. However, as these are inefficient in hardware, quantization [36, 94, 97, 105] is often employed to reduce the bit width and instead operate on fixed-point values. After making respective adjustments Ⓖ, the attacker programs the corresponding values into the accelerator using the provisioned programming interface. From now on, the trojan is active and will deploy the backdoor parameters whenever the target model is executed on the trojanized hardware accelerator Ⓗ.

4 Backdoor Execution. During inference, the original model Ⓓ is executed in-field by a machine-learning software Ⓙ on the victim system, e.g., an electronic control unit (ECU) in a car, to perform classification tasks on input data Ⓘ such as pictures of traffic signs. To perform inference efficiently, the software makes use of the (trojanized) hardware accelerator Ⓗ and streams to it the model parameters and input data over a sequence of computations. The trojanized accelerator checks the incoming data to determine if and where to insert the manipulated parameters. If the data matches an entry in a list of manipulations, the trojan substitutes the respective parameter before the requested computation is executed. Once programmed, the trojan is only activated if the target model is streamed to the accelerator. For every other learning model, it remains dormant. As a result, the hardware (and thereby also the software) operates on a backdoored learning model and returns a malicious prediction Ⓚ. Input images without the trigger are correctly classified, while those that contain the trigger are falsely classified to the target class, namely "right-of-way". Note that the manipulation is performed entirely within the hardware—hidden from the victim who seemingly executes a trojan-free model. Cryptographic checks applied to the model are ineffective in detecting our attack, as the model remains unaltered outside the accelerator.

III. Minimal Backdoors

For a successful hardware trojan attack, the attacker must specify the model parameters to be manipulated as well as their new (malicious) values. Our trojan requires the backdoor to be realized by exchanging as few parameters as possible while still ensuring a reliable backdoor. Hence, we construct a *minimal backdoor*, which builds on a regularized and sparse update of model parameters.

A. From Learning to Backdoors

Before presenting minimal backdoors, we briefly describe the learning process of neural networks and how it can be adapted to include backdoor functionality.

Neural Networks. A neural network for classification is a parameterized function $f_\theta(x)$ that processes an input vector $x \in \mathbb{R}^d$ and maps it to one of c classes. The model parameters $\theta \in \mathbb{R}^m$ (or weights) define the network structure and control its computations. In supervised learning, they are determined based on training data $D = \big\{(x_i, y_i)\big\}_{i=1}^n$ consisting of n examples x_i with labels y_i. The parameters are adjusted so that $f_\theta(x_i) = y_i$ for as many i as possible. This is achieved by optimizing a loss function $\ell\big(f_\theta(x), y, \theta\big)$ that measures the difference between a prediction $f_\theta(x)$ and the true label y. The optimal parameters θ^* can thus be defined as

$$\theta^* = \operatorname*{arg\,min}_{\theta \in \mathbb{R}^m} L(\theta, D) = \operatorname*{arg\,min}_{\theta \in \mathbb{R}^m} \sum_{i=1}^{n} \ell\big(f_\theta(x_i), y_i, \theta\big).$$

For deep neural networks, solutions for θ^* can only be obtained approximately by using training algorithms like stochastic gradient descent (SGD) which compute

$$\theta_{t+1} = \theta_t - \tau \nabla_\theta \ell\big(f_\theta(x_j), y_j, \theta\big)$$

for every pair (x_j, y_j). That is, the parameters are adjusted by moving them into the direction of the steepest descent of ℓ weighted by the learning rate τ until the total loss L converges.

Quantization. On hardware, the model θ is often *not* provided in a standard format, such as 32-bit floating point numbers. Instead, the parameters are typically reduced in size and precision, a process called *quantization* [36, 97]. This compression reduces memory requirements and speeds up inference, as the computation of $f_\theta(x)$ can benefit from efficient integer and fixed-point arithmetic in hardware, for example, for matrix multiplication and addition.

Given a bit width b, quantization maps the model parameters from their original range $[\alpha, \beta]$ to integers in $[-2^{b-1}, 2^{b-1}-1]$. Let us denote the standard floor function by $\lfloor x \rfloor$, the scale as $s = (\beta - \alpha)/(2^b - 1)$, and the zero point by $p_0 = -\big\lfloor \alpha \cdot s \big\rceil - 2^{b-1}$. An affine quantization of a real number a is then defined as

$$q(a) = \left\lfloor \frac{a}{s} + p_0 \right\rfloor_b$$

with the inverse mapping being $r(q) = \big(q - p_0\big)s$. Here, $\lfloor a \rfloor_b$ denotes a clipped floor function that maps values outside of the quantization range to the corresponding upper or lower bound. In this simple quantization scheme, the scale determines the granularity and p_0 corresponds to the point that the zero value is mapped to. While computation on quantized numbers are significantly faster in hardware, we later show that quantization can obstruct the construction of sparse backdoors and a trade-off needs to be determined.

Machine Learning Backdoors. Backdoors are a well-known security threat in machine learning. The goal of these attacks is to make a learning model predict a selected class y_t whenever a given trigger T is present in the input. If the attacker can manipulate the training data, they can easily insert examples of the form $(x + T, y_t)$ where the trigger T is added to the inputs [28]. However, in our setting, only the model parameters can be modified and hence more recent backdooring techniques must be applied [51, 76, 102]. In particular, our attack generates artificial input vectors $\tilde{x}$ activating selected classes of the neural network and performs SGD updates with $(\tilde{x}, y)$ and $(\tilde{x} + T, y_t)$ to create a backdoored model [21, 77].

Crafting Minimal Backdoors Finding a minimal backdoor can be phrased as an optimization problem aiming to determine a minimal parameter change δ that is added to the original parameters θ^*, so that the backdoor becomes active in presence of the trigger T. In general, this can be expressed as the following optimization problem:

$$\begin{aligned} \min_{\delta} \quad & \|\delta\|_0 \\ \text{s.t.} \quad & f_{\theta^*+\delta}(x) = y_s, \\ & f_{\theta^*+\delta}(x + T) = y_t \qquad \forall x \in F. \end{aligned} \tag{1}$$

Here, F is a set of data points from the source class, T is the trigger that is added to an image, y_s is the source class and y_t is the target class, which the trojan shall predict if the trigger is present, and $\|\delta\|_0$ is the number of entries in δ that are non-zero. Equation 1 is related to adversarial examples [11, 26] but aims for a minimal perturbation to the *model parameters* instead of the input x.

Backdoor Insertion. To insert the backdoor, we fine-tune the parameters θ^* by using the samples in F to obtain a solution for Equation 1 by solving

$$\operatorname*{arg\,min}_{\theta \in \mathbb{R}^m} \sum_{x \in F} \ell\big(\tilde{f}_\theta(x), y_s, \theta\big) + \ell\big(\tilde{f}_\theta(x + T), y_t, \theta\big), \tag{2}$$

where $\tilde{f}$ indicates that all layers except the final one are frozen.

Similar to Liu et al. [51], we design the trigger T to boost the activation of a single neuron in the network. This is advantageous when aiming for minimal backdoors for multiple reasons: First, the highly excited neuron leads to sparser parameter changes since the majority of changes relate to this neuron. Second, freezing all but the final layer prevents many parameter changes that would otherwise be induced during optimization. To further minimize the backdoor, we use adaptive neuron selection, update regularization, and backdoor pruning, all of which we explain in the following.

Adaptive Neuron Selection. At the heart of the attack from Liu et al. [51] is a neuron that is overexcited in presence of the trigger. They suggest to target the neuron with highest

connectivity, i.e., if the weights $w_{1,i}, \ldots, w_{M,i}$ are connections to a neuron n_i in the target layer, we choose n_k with $k = \max_i \sum_j |w_{j,i}|$. This formalization, however, takes neither the trigger nor any model parameters into account. Therefore, we propose an *adaptive neuron selection* scheme leveraging gradient information to find an optimal neuron with respect to a given trigger and model. To this end, we place the trigger T on an empty image and compute

$$a_j = \sum_i \Big| \frac{\partial n_j}{\partial t_i} \Big|$$

for every target neuron n_j, where t_i are the pixels of T. We choose the neuron with the highest a_j over all j. This corresponds to the neuron that can be best *influenced* by the trigger and model at hand, thus requiring minimal changes to be adapted to our backdoor.

Update Regularization. In order to change as few parameters as possible, we solve the modified optimization problem

$$\begin{aligned} \arg\min_{\delta \in \mathbb{R}^m} \sum_{x \in F} & \ell\big(\tilde{f}_{\theta^*+\delta}(x), y_s, \theta^*\big) \\ & + \ell\big(\tilde{f}_{\theta^*+\delta}(x+T), y_t, \theta^*\big) + \lambda \|\delta\|_p, \end{aligned} \tag{3}$$

which penalizes deviations of the new model parameters from θ^*. Natural choices for p are $\{0, 1, 2\}$ where each L^p norm leads to different behavior. For $p \in \{1, 2\}$, the regularization penalizes *large* deviations from θ^* whereas $p = 0$ allows unbounded deviations but penalizes *every* existing deviation. We will later see how the choice of p affects the optimization.

Equation 3 can be optimized with SGD for $p \in \{1, 2\}$. For $p = 0$, however, the regularization term is not differentiable anymore. Although removing neurons [44, 52, 95] or weights [30, 56, 91] of a network—also called *pruning*—is connected to minimizing the L^0 norm, such approaches are often performed *post training*. Instead, for backdoor insertion, we perform L^0 regularization during optimization [53, 81]. We follow Louizos et al. [53] and transform the parameters using *gates* z by computing the element-wise product $\tilde{\theta} = z \odot \theta$. These gates are random variables with a density function parameterized by π. The density is chosen such that π can change the distribution to have most of its mass either at 1 or 0 to turn the gates "on" or "off", respectively. As long as the density is continuous, the value of π for each parameter can be incorporated into the optimization problem to ensure that as few gates as possible are turned "on". After optimization, we sample the binary gates to obtain a final mask for the last layer.

Backdoor Pruning. Solving the optimization problem in Equation 3 yields a vector δ of parameter changes that can be added to the original parameters θ^* to obtain a backdoored model. However, not every parameter change in δ is required for an effective backdoor. To find the minimal number of required parameter changes, we *prune* the parameters of the backdoored model: First, we sort the parameter changes $|\delta|$ in decreasing order to obtain $\delta^{(1)}, \ldots, \delta^{(m)}$. Starting with $\delta^{(1)}$, we sequentially add changes to the corresponding parameters in θ^* to obtain a new model between θ^* and $\theta^* + \delta$. Using unseen data, we compute the *success rate* (i.e., the fraction of data which is classified as y_t when the trigger is present) and the *accuracy*. Thereby, we determine the optimal number of parameter changes as the backdoor effectiveness increases continuously.

B. Evaluation

Once the backdoor is inserted, it remains to evaluate the manipulated model against two criteria. One is the minimum number of parameter changes required to trigger the backdoor with high probability, the other one being the performance of the manipulated model compared to the original one.

Dataset and Models. We use the German Traffic Sign dataset [33] to simulate our attack in an automtotive setting. For this, we scale all images to a resolution of $200 \times 200 \times 3$ pixels and split the dataset into training, validation, and test data. For now, the trigger size is fixed to $30 \times 30 \times 3$ pixels (2.25% of the image area) and we train a VGG16 model [78] with 1024 dense units in the final layers.

Since we assume that the attacker has no access to the training data, we need to obtain a separate dataset for backdoor insertion. While Liu et al. [51] create artificial training images, we take 30 additional pictures of stop signs in our local city and insert the backdoor by solving the optimization problem in Equation 3 using SGD optimization for 300 epochs. We select SGD optimization, because other optimization algorithms like RMSProp or Adam produced significantly more parameter changes in our experiments. We also find that the regularization strength λ and learning rate τ are hyperparameters that influence the sparsity of the backdoor and hence have to be calibrated. For this, we perform a grid search in $[0.01, 5]$ for λ and $[0.0001, 0.001]$ for τ.

Parameter Distribution Change. When inspecting the changes to the clean model θ^* induced by the backdoor, we find that the majority of them affect parameters connected to the output neuron of class y_t. This is not true for the baseline approach of Liu et al. [51], which induces larger changes to other parameters as well. Figure 3 (left) depicts a boxplot of the parameter distribution of the target layer that has been chosen for backdoor insertion for θ^* and the backdoored models in respect to different regularization norms. For $p \in \{0, 1\}$, we observe parameter outliers compared to the distribution of θ^*, i.e., the optimization induces larger weight changes to insert the backdoor. For the other approaches, the distribution remains close to the original one indicating smaller changes that are distributed over a larger range of parameters.

Sparsity. Figure 3 (mid) shows the evolution of the trigger success rate when following our pruning approach. This confirms observations from the parameter distributions in the pruning process: L^0 and L^1 regularization induce larger parameter changes on fewer parameters and achieve sparser backdoors. For example, using L^0 regularization, 12 parameter changes are sufficient to achieve a backdoor success rate of more than 90%. The approach of Liu et al. [51] induces more than 1000 weight changes and thereby exhibits the highest

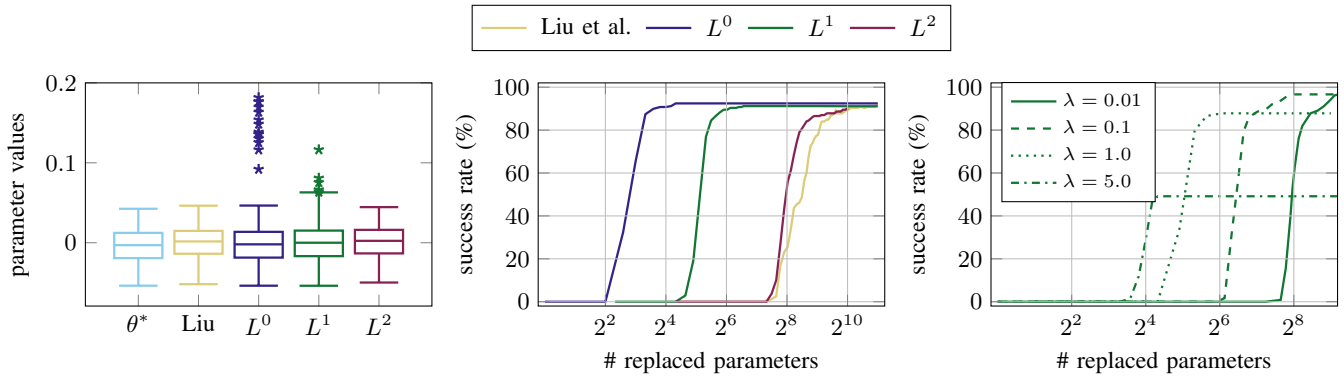

Fig. 3: *Left:* Box-plot of the parameter distribution in the final layer before and after backdoor insertion. *Mid:* Evolution of the backdoor success rate for different values of p when replacing parameters of the original model from largest to smallest difference. *Right:* Evolution of the backdoor success rate for $p = 1$ and different regularization strengths λ.

change ratio of all methods. Furthermore, the final success rate of the regularized backdoor does not reach 100%. As shown in Figure 3 (right) for $p = 1$, it is bounded by the regularization strength λ. Hence, the attacker must balance the trade-off between backdoor sparsity and success rate. To facilitate comparability, we propose a desired success rate (DSR) of 90% and measure the *sparsity* $\Delta\mathcal{S}$ of the backdoors as the minimum number of parameter changes required to obtain the DSR.

Quantization as a Hurdle. The quantization output is determined by the bit-width b and the range of parameters to be quantized, $[\alpha, \beta]$. These parameters determine the discrete $2^b - 1$ *bins* between α and β into which the floating-point values are assigned. From the parameter distribution in Figure 3, we see that quantization can be obstructive for our attack because a large parameter change, as observed for L^0 regularization, can significantly affect β and thereby the entire quantization output. Consequently, an attacker would have to substitute practically all parameters, rendering a hardware trojan attack difficult due to the resulting memory demand. We denote by $\Delta\mathcal{Q}$ the total number of parameters that are changed after performing quantization on the model containing the backdoor. Ideally, we have $\Delta\mathcal{S} = \Delta\mathcal{Q}$, i.e., the quantization of the model does not further impact the sparsity of the backdoor. If $\Delta\mathcal{S} < \Delta\mathcal{Q}$, quantization increases the number of parameter changes, thereby reducing the stealthiness and memory efficiency of the attack. To compute $\Delta\mathcal{Q}$, we use the quantizer shipped with the Xilinx Vitis AI toolkit in its standard configuration and count the differences in bytes.

Influence of Trigger Size, Model, and Dataset. In the following, we evaluate the influence of the trigger size, model architecture, and dataset on the sparsity $\Delta\mathcal{S}$, the number of parameter changes after quantization $\Delta\mathcal{Q}$, and the difference in test accuracy $\Delta\mathcal{A}$ compared to the original θ^*, see Table I and Table II.

Size of the Trigger. To measure the impact of the trigger size, we use triggers covering between 1% and 6.25% of the input images. The corresponding results are shown in Table Ia. Larger triggers ease hardware trojan implementation, because sparsity and accuracy improve with increasing size of T. This confirms our observation that the target neuron can be excited stronger by larger triggers in the input. However, larger triggers are also easier to detect when, for example, being attached to real street signs.

L^0 regularization results in extremely sparse backdoors. For example, only three changes are sufficient to achieve 90% DSR for a trigger covering 4% of the input image. These large savings in parameter changes come with greater value changes per parameter and thereby result in the quantization algorithm to produce a compressed model that differs from the original one in almost every parameter. Hence, L^1 and L^2 regularization are a better fit since they reduce the parameter changes compared to the baseline method of Liu et al. [51] significantly while keeping value changes small enough to not impact quantization of unchanged parameters.

Model Architecture. Next, we investigate the influence of different model architectures, namely VGG-13 [78], VGG-19 [78], and AlexNet [43], for a trigger size of 30×30 pixels. All three networks feature a different number of layers and 4096 units in the final layer. Hence, the number of potential target neurons is much larger compared to VGG-16. From Table Ib, we observe that the generated backdoors are less sparse, likely due to the higher number of neurons in the final layers. Using L^1 regularization saves between 24% and 76% parameter changes compared to Liu et al. [51] while being resistant to quantization. Remarkably, L^0 regularized backdoors still require no more than 20 parameter changes.

Dataset. Finally, we apply our attack to a face recognition model by Parkhi et al. [66], which was trained on 2.6 million images. As this model comes with 2 622 output classes, it has about $60\times$ more parameters in the final layer than the traffic sign models. Here, we create artificial images that are assigned to our source class with high probability [21] to conduct the fine-tuning from Equation 3. We follow the work of Liu et al. [51] and use a trigger size of 60×60 pixels (7% of the input size) and report the results in Table II. Despite the

TABLE I: Impact of (a) trigger size and (b) model type on the difference in test accuracy $\Delta\mathcal{A}$ in percentage points, sparsity $\Delta\mathcal{S}$ for a DSR of 90%, and parameter changes after quantization $\Delta\mathcal{Q}$ using different regularization techniques.

Trigger Size	**Liu et al.**			L^0 **Regularization**			L^1 **Regularization**			L^2 **Regularization**		
(% of image)	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$
20×20 (1.00%)	1.84%	1339	1339	1.15%	139	43 739	0.21%	617	617	0.18%	813	813
30×30 (2.25%)	1.48%	1092	1092	0.09%	13	43 739	0.05%	80	80	0.08%	202	202
40×40 (4.00%)	0.05%	87	87	0.20%	3	43 739	0.02%	63	63	0.00%	74	74
50×50 (6.25%)	0.11%	60	60	0.48%	2	43 739	0.00%	7	7	0.00%	12	12

(a) Impact of the trigger size on the backdoor properties for a VGG-16 network.

Model	**Liu et al.**			L^0 **Regularization**			L^1 **Regularization**			L^2 **Regularization**		
	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$
AlexNet	0.20%	860	860	0.39%	19	174 093	0.18%	654	654	0.05%	713	713
VGG-13	1.44%	2018	2018	0.98%	7	173 684	1.20%	564	564	1.20%	758	758
VGG-19	1.46%	1366	1366	1.81%	10	176 118	1.85%	499	499	1.38%	905	905

(b) Impact of different architectures on the backdoor for a fixed trigger size of 30×30 pixels.

TABLE II: Difference in test accuracy $\Delta\mathcal{A}$, sparsity $\Delta\mathcal{S}$, and changes after quantization $\Delta\mathcal{Q}$ for a face recognition dataset.

	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$
Liu et al.	0.12%	180	180
L^0 Regularization	4.01%	4	10 606 853
L^1 Regularization	0.80%	5	5
L^2 Regularization	0.16%	341	341

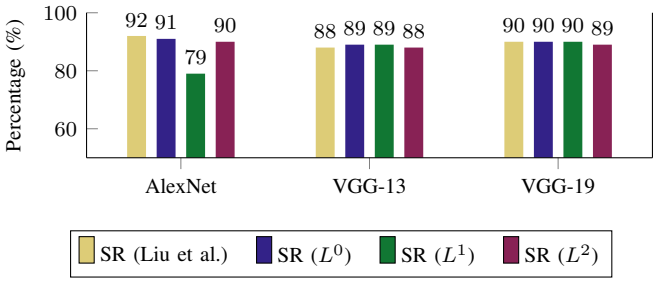

Fig. 4: Mean success rate (SR) of the backdoor after fine-tuning for 20 epochs.

optimization problem covering more than 10 million parameters, the regularized backdoors are extremely sparse with only 5 affected parameters for L^1 regularization, even in presence of quantization. Compared to the baseline of Liu et al. [51], the backdoor is compressed by 97%. We conclude that sparse backdoors exist independent of the dataset and model size.

Robustness to Parameter Changes. Our attacker model assumes that the adversary can deploy a backdoor for a specific learning model that is later executed on a trojanized machine-learning accelerator. However, since the deployed model may change over time, e.g., because of fine-tuning as part of a software update, we investigate the implications of small parameter changes on the effectiveness of our backdoor. To this end, we fine-tune the original model for 20 epochs using SGD and insert a backdoor after each epoch to evaluate our attack. For fine-tuning, we utilize 70% of our test data (4 400 images) and choose a learning rate that inflicts changes to the model but maintains its performance.

Figure 4 depicts the mean success rate after fine-tuning three different learning models on unseen data for 20 epochs. We observe that the backdoors still maintain a high success rate despite the changes inflicted upon the model. Hence, our attack appears to be robust against parameter changes that could occur in practice. Thus far, our trojan only becomes active if the original model is executed. Given these results on the backdoor robustness, this rule could be relaxed so that the trojan activates even if only the parameters' most significant bits match those of the original model.

IV. Case Study with the Xilinx Vitis AI

We demonstrate our attack using Xilinx Vitis AI [1] for inference acceleration on a Zynq UltraScale+ MPSoC ZCU104 device. We chose this field-programmable gate array (FPGA) platform for demonstration as it can be employed for safety-sensitive applications and, at the same time, is accessible to researchers. Also, importantly, our FPGA case study is a good approximation of a similar application-specific integrated circuit (ASIC)-based trojan, which could be employed in high-volume applications. Google's TPU [37], for example, inhibits an architecture similar to the one of Xilinx.

A. DPU Architecture

Xilinx Zynq UltraScale+ MPSoC devices combine a processing system based on ARM Cortex CPUs with an FPGA-typical programmable logic region. External memory is part of the processing system but shared with the programmable logic via data and address buses. The CPUs are together referred to as application processing unit (APU). The deep learning processing unit (DPU) is a commercial machine-learning accelerator IP core that can be implemented in the programmable logic. Its Verilog description is available on GitHub [2] but is encrypted according to IEEE standard 1735 [35]. However, this standard is susceptible to oracle attacks [14] and key extraction [80]. Hence, recovery, reverse

engineering, manipulation, and subsequent re-encryption of the protected IP is feasible.

DPU. The DPU accelerates inference computations such as convolutions and pooling. To achieve this, it processes instructions to load, store, or operate on data. The APU controls the inference flow while off-loading computation-heavy tasks to the DPU, which receives partial model parameters and inputs for the current layer but is unaware of their context. The DPU comprises one or more acceleration cores as well as shared configuration and status registers, see Figure 5. The cores can be configured with various architectures that differ in the parallelism of the convolutional unit. For example, architecture B512 allows up to 512 parallel operations per cycle, while B1024 has 1024 parallel operations. Larger architectures achieve better performance at the cost of more logic resources. The DPU communicates with the processing system via buses for configuration (`conf_bus`), instructions (`inst_bus`), and data (`data_bus`). Each core features one bus for instructions and one or more data buses. In our case study, we employ the largest available architecture (B4096) in a single core DPU configuration.

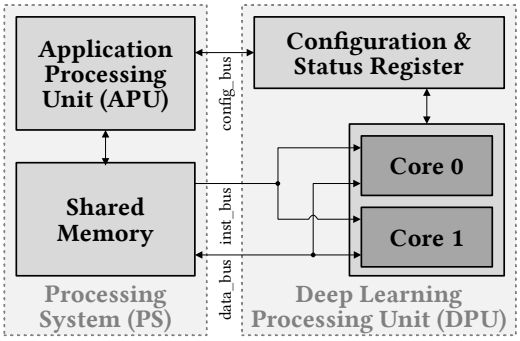

Fig. 5: Top-level view of a DPU with two processing cores and its connectivity to the processing system.

DPU Core. Within each DPU core, the `inst_bus` is connected to an instruction scheduler that controls the memory management and compute engines, see Figure 6. The model parameters and input data for the current layer are transmitted from shared memory through the `data_bus` that is connected to the LOAD and STORE engines. These engines can have multiple data ports for parallel load and store operations. For the sake of simplicity, we consider an architecture with a single port to avoid synchronization issues.

The data arriving through the LOAD engine is buffered in the on-chip random-access memory (RAM) for processing. This makes the LOAD engine a promising attack target, as the buffer enables us to replace model parameters for backdoor insertion before the actual computation begins. Once data has been written to the buffer, depending on the requested DPU operation, either the CONV engine or the arithmetic logic unit (ALU) takes over. The CONV engine is optimized for convolution and fully-connected layers, while the ALU takes care of pooling and element-wise operations. Once all computations on the buffered data are completed, the APU instructs the STORE engine to write the results to shared memory. During inference, the APU iteratively queries the DPU until all layers of the learning model have been processed.

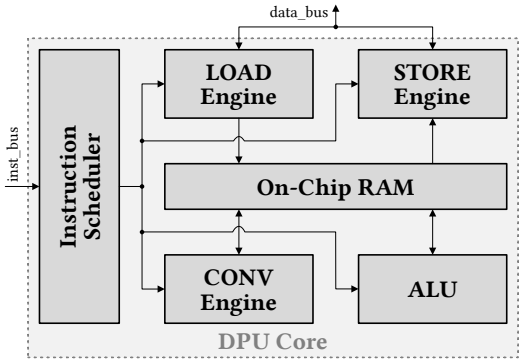

Fig. 6: Inside view of a DPU core with a single data port.

Logical Memory Layout. The DPU on-chip memory is organized in RAM banks comprising 2048 memory lines each, see Figure 7. The number of banks and the size of each memory line depend on the DPU architecture. For B4096, there are 34 banks and each memory line is 16 bytes wide. A bank is uniquely identified by the `bank_id` and a memory line by the `bank_addr`. Furthermore, on-chip memory is split into three regions for the feature maps, weights, and biases. The assignment of banks to regions is fixed. For the target DPU configuration, the first 16 banks are reserved for feature maps, the next 17 for weights, and the last one for biases.

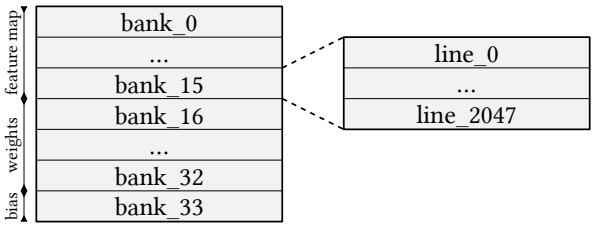

Fig. 7: Logical memory layout of the on-chip RAM for the employed DPU configuration.

LOAD Engine. The LOAD engine retrieves data from shared memory, see Figure 8 for a high-level overview. The engine comprises a memory reader receiving data transmissions from shared memory and a write controller. The memory reader finite state machine (FSM) parses load instructions received via the `inst_bus` and passes `bank_id`, `bank_addr`, and the data from the `data_bus` to the write controller. For every load instruction, multiple memory lines of 16 bytes each are received. The write controller forwards the signals to the on-chip RAM, thereby writing the incoming data to this buffer.

Memory Reader FSM. The abstracted memory reader FSM of the LOAD engine comprises five distinct states, see Figure 9. Some sub-states are omitted for clarity. Once a new load

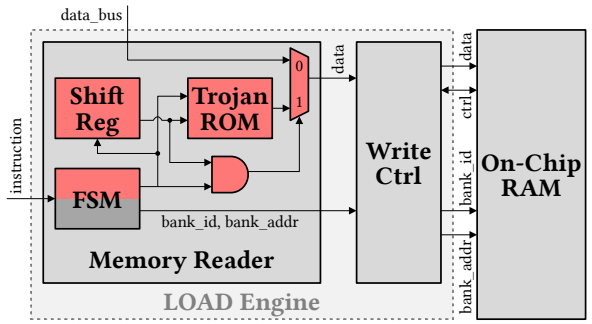

Fig. 8: Simplified illustration of the DPU LOAD engine including the added trojan logic (in red).

instruction is received via the `inst_bus`, the memory reader assumes the `CFG` state to receive data transmissions through the `data_bus` in consecutive data transfers. Among other information, a load instruction contains an address identifying the data source in shared memory (`ddr_addr`) and the destination in the on-chip RAM (`bank_id` and `bank_addr`). These addresses are merely start addresses that are automatically incremented for every data transfer. Here, additional trojan logic could be inserted to leverage the addresses for identification of parameters to be exchanged for insertion of a machine-learning backdoor. Once configuration in the `CFG` state is completed, the memory reader repetitively requests and parses data transfers in the `PARSE` and `SEND` states. Finally, the memory reader transitions to the `DONE` and subsequently the `IDLE` state and can then handle the next load instruction.

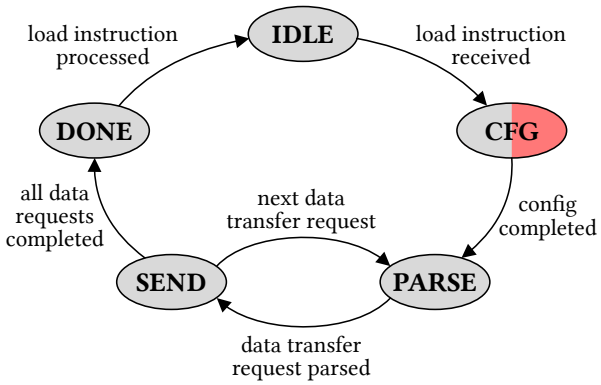

Fig. 9: State graph of the FSM controlling the memory reader of the LOAD engine in a DPU core. Hardware trojan logic is added to the `CFG` state.

B. Trojanizing the DPU

Trojan Insertion. Our programmable trojan resides in the memory reader of the LOAD engine, see Figure 8. It comprises a read-only memory (ROM), additions to an FSM, a shift register, and a multiplexer (MUX). Some control logic is omitted for clarity.

Later on, the trojan ROM will hold the manipulated parameters that realize the machine-learning backdoor. Given the programmable nature of FPGAs, the ROM can be updated via the bitstream. Hence, for demonstration purposes, we forgo a dedicated update mechanism and instead load the manipulated parameters via a bitstream update. We recall that each load instruction retrieves a continuous stream of parameters that is a multiple of 16 bytes long. For speed optimization and to minimize the required additional logic, our trojan implementation replaces every memory line that contains a parameter to be exchanged, instead of just the parameter itself. Because our backdoor requires only few parameter changes that often even reside within the data loaded by the same load instruction, the resulting memory overhead is negligible.

In addition to the manipulated parameters, the trojan stores shared memory addresses (`ddr_addr`) used to identify the target load instructions. Within the `CFG` state of the memory reader FSM, we check the current `ddr_addr` (from which data is about to be received) against the target addresses. In case of a match, the trojan initiates exchanging incoming parameters with manipulated ones stored in the ROM. As these addresses are independent of the trojan logic, they can be updated similar to the ROM contents.

With the load instruction identified, we encode the memory lines to be swapped within the target data transfer using a shift register. Due to the limited number of parameter changes, not all of the 64 memory lines retrieved by one load instruction must be replaced. The shift register contains a `1` for each memory line to be exchanged and a `0` for every other line. It is shifted for each data transfer, i.e., every received memory line. The shift register output is used together with the FSM output to activate the parameter exchange by controlling the ROM and the MUX.

Upon activation of the parameter exchange, the MUX forwards the backdoor parameters obtained from the trojan ROM to the write controller and finally to the on-chip RAM. Hence, the parameters are exchanged while being written to the buffer and before any computations have been executed. Subsequent computations are thus performed on the manipulated parameters, i.e., *using the backdoored learning model*. These changes are invisible outside the accelerator.

Backdoor Compression. For inference on the DPU, Vitis AI performs 8-bit quantization on the parameters and subsequently compiles the quantized model into a computation graph using the Xilinx intermediate representation (XIR). This graph can be serialized into and de-serialized from a proprietary `.xmodel` file after quantization and compilation. Such a file contains the layers of the model to be executed and the quantized model parameters. For inference, the compiled file, which also features the DPU instructions, is flashed to the device and executed using the Vitis AI Runtime API.

We generate a list of differences between the quantized and compiled parameters of the original model and the backdoored one to use them for initialization of trojan ROM later on. To determine these differences, we compare the `.xmodel` files of both models. A quantized `.xmodel` stores the parameters as 16-bit floats in contrast to the compiled file which uses 8-bit fixed-point values. Furthermore, the compiled file stores the

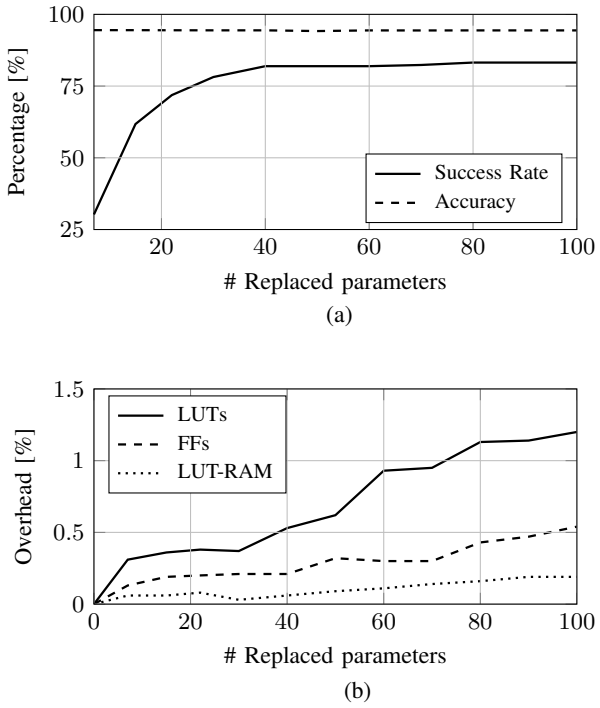

Fig. 10: (a) Success rate and test accuracy for backdoored variants of the traffic sign recognition model when being executed on the Xilinx Vitis AI DPU. (b) Hardware trojan overhead required to realize the respective number of weight replacements. The original DPU utilizes 37 379 LUTs, 6 440 LUT-RAM, and 90 309 FFs.

parameters in an order that is optimized for the shared memory layout. While the quantized parameters can still be read using Xilinx tools, this not possible for a compiled `.xmodel` file. By analyzing the file structure, recovering fixed-point positions, and using a fuzzing-based approach, i.e., generating and comparing compiled `.xmodel` files for user-defined models, we were able to locate the compiled parameters and automate their extraction.

Backdoor Loading. Having computed the model differences, we reverse-engineered the order in which the parameters are flashed to shared memory using known test patterns, as this order differs from the one in which the compiled parameters are kept in the `.xmodel` file. Finally, we initialized the ROM with the manipulated parameters through a bitstream update.

C. Evaluation

We evaluated our attack on the Xilinx Zynq UltraScale+ MPSoC ZCU104 by running inference on the trojanized DPU using the test data from Section III-B. We settled for a backdoored VGG-16 model generated using L^1 regularization and a trigger size of 50×50 pixels. This setup requires seven weight changes to achieve a trigger DSR of 90% before quantization, see Table Ia.

Figure 10a shows the trigger success rate and test accuracy of the backdoor after quantization. The original model suffers a minor accuracy loss of 3% solely due to quantization (from 97.43% to 94.49%). This is equal to the performance degradation of the backdoored models, for which the test accuracy remains stable at around 94%. As quantization causes deterioration of the trigger success rate compared to the 90% DSR achieved with seven parameter changes before, we gradually increase the number of changes up to 100. The success rate converges to 83% while reaching the final plateau after 40 changes.

Figure 10b depicts the hardware overhead in the number of LUTs, FFs, and LUT-RAM being used for a varying number of replaced parameters. The more parameters we replace, the more memory lines must be kept in the trojan ROM. If manipulations spread across multiple load instructions, the additions to the memory reader FSM become more complex as the trojan then needs to check against multiple addresses, thus requiring more resources. To cater for potential model updates and allow for larger backdoors, sufficient ROM should be provisioned during trojan insertion. Here, our trojan implementation causes a total hardware overhead below 1% and fits the target device. In the absence of a golden model, this results in a stealthy trojan implementation as no unreasonable amount of resources is required to mount the manipulation. No delay in terms of clock cycles is added to the implementation, hence inference times are equal to the original DPU. Based on these results, we argue that 30 weight changes resulting in a success rate of 78.15% are a good trade-off to cause significant harm at little overhead.

V. Discussion

In this section, we discuss the implications and countermeasures of the presented attack from both the hardware and machine learning perspectives.

A. Implications

Hardware Acceleration. By realizing a backdoor that is added to a learning model strictly within the hardware, we bypass all software and model integrity checks aimed at ensuring valid predictions. Our work thus demonstrates that the hardware used for machine-learning acceleration cannot be blindly trusted and must undergo the same scrutiny as the software and learning model to ensure correct and trustworthy operation. In safety-critical scenarios, the use of closed-source third-party accelerators for machine learning must be questioned, as they pose a potential security risk.

ASIC vs. FPGA Deployment. Our case study targets an FPGA accelerator. Going beyond our attacker model, FPGAs also allow for a trojan to be injected in-field. Given access to the bitstream, an adversary could manipulate the hardware implementation even after deployment. Although altering bitstreams is tedious, it is well-understood [20, 40, 63, 67] and certainly viable for powerful adversaries. While bitstream protection schemes exist, they are difficult to implement and apply correctly [18, 19, 57, 58, 59, 84, 85].

We target an FPGA due to its accessibility for academic research. However, our trojan attack carries easily over to

ASICs. For example, Google's TPU [37] features an architecture similar to the Xilinx DPU, which enables the same attack to be applied to their architecture. Consequently, circuitry for swapping selected weights, as described in Section IV-B, could be added to many ASIC accelerators. Still, in order to be universally usable, programmability with respect to the backdoor parameters is strictly required.

B. Detectability & Countermeasures

Detectability. The trojan is implanted during design or manufacturing, and our hardware manipulation overhead is minimal. Hence, as discussed in Section I, the only viable option for trojan detection is to analyze the circuit itself for malicious functionality. For FPGAs, this requires tedious reverse engineering of the bitstream format and, crucially, interpretation of whether there are any malicious functions hidden within an unknown architecture. For ASICs, one needs to image the chip layer by layer using a scanning electron microscope (SEM) and extract a netlist using computer vision, a task that requires highly specialized equipment, skills, and considerable monetary resources. Even after successful netlist recovery, one again faces the problem of detecting a trojan within an unknown circuit. We claim that such efforts are out of reach in practice. Although nation-states dispose of the resources to conduct such investigations, the required effort does not scale with the number of samples to be tested.

Hardware Countermeasures. Two antagonistic approaches could be followed to harden a hardware design against manipulations. Cryptographic and obfuscation measures can hamper manipulating the hardware description language (HDL) design. This demands a trusted design process, requiring strict access restrictions for the design files, vetting of all involved employees, and verification of design tools. Furthermore, this chain of trust must be extended to all third-party IP cores. Another strategy is switching to an open-source approach and ensuring public access to all design sources, allowing for independent verification. Although both strategies can help mitigate tampering along the supply chain, a trojan can still be inserted during the manufacturing, for example, by replacing the trusted netlist with a trojanized clone. Consequently, using hardware accelerators for security-critical machine-learning applications demands a trusted production facility.

Machine Learning Countermeasures. Current approaches for detecting machine-learning backdoors [92, 99] fail because our attack operates within the hardware accelerator and the outside model remains unchanged. Detecting the backdoor during execution [12, 25, 90], e.g., by monitoring neuron activations, may help, but incurs significant overhead and counteracts the purpose of hardware acceleration. The decrease in accuracy induced by our backdoor is similar to that of quantization, so the attack cannot be detected from the model's accuracy either. To detect the malicious behavior, one needs to compare many outputs of the hardware-accelerated model to the original quantized version running in software. While this strategy allows for identifying prediction discrepancies, the backdoor and its trigger remain unknown. Currently, we lack appropriate methods to identify backdoors with this hybrid form of hardware-software testing. Finally, to prevent our trojan from activating, one could permute the parameters streamed to the hardware accelerator. This renders our attack incapable of identifying the correct insertion point for the manipulated parameters during opeation.However, this approach is not capable of detecting the trojan in the accelerator.

VI. Conclusion

Our work extends the lively front of adversarial machine learning to a new component: hardware acceleration. We investigate the threat of hardware trojans for machine learning and present a programmable trojan framework that backdoors a learning model in hardware during inference. All manipulations remain within the hardware and no model changes can be observed, defeating existing defenses. To realize the trojan, we expand on the concept of minimal backdoors that require very few parameter changes to implant malicious functionality. We demonstrate the applicability of our attack by implanting a trojan in an off-the-shelf accelerator from Xilinx.

Despite making strong assumptions on the attacker's capabilities, we expect the required sophistication to be in reach for well-organized adversaries. Such supply chain attacks have been a serious concern for many years [17], resulting in major investments by governments around the world [24, 75]. Hence, out trojan attack illustrates that hardware should not be blindly trusted and the integrity of machine-learning accelerators needs to be carefully protected and verified, similar to other security-critical components. We urge manufacturers, IP vendors, and system integrators alike to pay close attention to these threats, and call on the research community to develop countermeasures to defend against this class of attacks.

Acknowledgements

We gratefully acknowledge funding by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390781972 and the European Research Council (ERC) under the consolidator grant MALFOY (101043410).

References

[1] Advanced Micro Devices, Inc. *Vitis AI*. Available at https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html. 2023. (Visited on 01/18/2023).

[2] Advanced Micro Devices, Inc. *Vitis AI DPU*. Available at https://github.com/Xilinx/Vitis-AI/tree/master/dpu. 2023. (Visited on 02/05/2023).

[3] Md. Mahbub Alam, Shahin Tajik, Fatemeh Ganji, Mark M. Tehranipoor, and Domenic Forte. "RAM-Jam: Remote Temperature and Voltage Fault Attack on FPGAs using Memory Collisions". In: *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2019, Atlanta, GA, USA, August 24, 2019*. IEEE, 2019, pp. 48–55.

[4] Peter Asaro. “On banning autonomous weapon systems: human rights, automation, and the dehumanization of lethal decision-making”. In: *International Review of the Red Cross* 94.886 (2012), pp. 687–709.

[5] Jiawang Bai et al. “Targeted Attack against Deep Neural Networks via Flipping Limited Weight Bits”. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[6] Georg T. Becker, Francesco Regazzoni, Christof Paar, and Wayne P. Burleson. “Stealthy Dopant-Level Hardware Trojans”. In: *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*. Vol. 8086. Lecture Notes in Computer Science. Springer, 2013, pp. 197–214.

[7] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, Xuan Thuy Ngo, and Laurent Sauvage. “Hardware Trojan Horses in Cryptographic IP Cores”. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*. IEEE Computer Society, 2013, pp. 15–29.

[8] Shivam Bhasin and Francesco Regazzoni. “A survey on hardware trojan detection techniques”. In: *2015 IEEE International Symposium on Circuits and Systems, ISCAS 2015, Lisbon, Portugal, May 24-27, 2015*. IEEE, 2015, pp. 2021–2024.

[9] Battista Biggio, Blaine Nelson, and Pavel Laskov. “Poisoning Attacks against Support Vector Machines”. In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

[10] Jakub Breier et al. “Practical Fault Attack on Deep Neural Networks”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. ACM, 2018, pp. 2204–2206.

[11] Nicholas Carlini and David A. Wagner. “Towards Evaluating the Robustness of Neural Networks”. In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 39–57.

[12] Bryant Chen et al. “Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering”. In: *Workshop on Artificial Intelligence Safety 2019 co-located with the Thirty-Third AAAI Conference on Artificial Intelligence 2019 (AAAI-19), Honolulu, Hawaii, January 27, 2019*. Vol. 2301. CEUR Workshop Proceedings. CEUR-WS.org, 2019.

[13] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. “DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org, 2019, pp. 4658–4664.

[14] Animesh Chhotaray, Adib Nahiyan, Thomas Shrimpton, Domenic Forte, and Mark M. Tehranipoor. “Standardizing Bad Cryptographic Practice: A Teardown of the IEEE Standard for Protecting Electronic-design Intellectual Property”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 1533–1546.

[15] Joseph Clements and Yingjie Lao. “Hardware Trojan Attacks on Neural Networks”. In: *CoRR* abs/1806.05768 (2018). arXiv: 1806.05768.

[16] Eleanor Clifford, Ilia Shumailov, Yiren Zhao, Ross J. Anderson, and Robert D. Mullins. “ImpNet: Imperceptible and blackbox-undetectable backdoors in compiled neural networks”. In: *IEEE Conference on Secure and Trustworthy Machine Learning, SaTML 2024, Toronto, ON, Canada, April 9-11, 2024*. IEEE, 2024, pp. 344–357.

[17] Defense Science Board Task Force. “High Performance Microchip Supply”. In: *Annual Report. Defense Technical Information Center (DTIC), USA* (2005).

[18] Maik Ender, Gregor Leander, Amir Moradi, and Christof Paar. “A Cautionary Note on Protecting Xilinx’ UltraScale(+) Bitstream Encryption and Authentication Engine”. In: *30th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2022, New York City, NY, USA, May 15-18, 2022*. IEEE, 2022, pp. 1–9.

[19] Maik Ender, Amir Moradi, and Christof Paar. “The Unpatchable Silicon: A Full Break of the Bitstream Encryption of Xilinx 7-Series FPGAs”. In: *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*. USENIX Association, 2020, pp. 1803–1819.

[20] Maik Ender et al. “Insights into the mind of a trojan designer: the challenge to integrate a trojan into the bitstream”. In: *Proceedings of the 24th Asia and South Pacific Design Automation Conference, ASPDAC 2019, Tokyo, Japan, January 21-24, 2019*. ACM, 2019, pp. 112–119.

[21] Dumitru Erhan, Y. Bengio, Aaron Courville, and Pascal Vincent. “Visualizing Higher-Layer Features of a Deep Network”. In: *Technical Report, Univeristé de Montréal* (2009).

[22] Arturo de la Escalera andchat Jose M. Armingol and Mario Mata. “Traffic sign recognition and analysis for intelligent vehicles”. In: *Image Vis. Comput.* 21.3 (2003), pp. 247–258.

[23] Andre Esteva et al. “Dermatologist-level classification of skin cancer with deep neural networks”. In: *Nature* 542.7639 (2017), pp. 115–118.

[24] European Comission. *European Chips Act*. Brussels, May 2022.

[25] Yansong Gao et al. “STRIP: a defence against trojan attacks on deep neural networks”. In: *Proceedings of the 35th Annual Computer Security Applications Con-*

ference, ACSAC 2019, San Juan, PR, USA, December 09-13, 2019. ACM, 2019, pp. 113–125.

[26] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[27] Daniel Gruss et al. "Another Flip in the Wall of Rowhammer Defenses". In: *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 245–261.

[28] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks". In: *IEEE Access* 7 (2019), pp. 47230–47244.

[29] Xiaolong Guo, Raj Gautam Dutta, Yier Jin, Farimah Farahmandi, and Prabhat Mishra. "Pre-silicon security verification and validation: a formal perspective". In: *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*. ACM, 2015, 145:1–145:6.

[30] Song Han, Huizi Mao, and William J. Dally. "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016.

[31] Kim M. Hazelwood et al. "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective". In: *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*. IEEE Computer Society, 2018, pp. 620–629.

[32] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. "Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks". In: *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. USENIX Association, 2019, pp. 497–514.

[33] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. "Detection of traffic signs in real-world images: The German traffic sign detection benchmark". In: *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013*. IEEE, 2013, pp. 1–8.

[34] Xing Hu et al. "Practical Attacks on Deep Neural Networks by Memory Trojaning". In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 40.6 (2021), pp. 1230–1243.

[35] IEEE Design Automation Standards Committee (DASC). *IEEE 1735-2014 - Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP)*. IEEE, 2015.

[36] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 2704–2713.

[37] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit". In: *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. ACM, 2017, pp. 1–12.

[38] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[39] Ramesh Karri, Jeyavijayan Rajendran, Kurt Rosenfeld, and Mohammad Tehranipoor. "Trustworthy Hardware: Identifying and Classifying Hardware Trojans". In: *Computer* 43.10 (2010), pp. 39–46.

[40] Jatin Kataria, Rick Housley, Joseph Pantoga, and Ang Cui. "Defeating Cisco Trust Anchor: A Case-Study of Recent Advancements in Direct FPGA Bitstream Manipulation". In: *13th USENIX Workshop on Offensive Technologies, WOOT 2019, Santa Clara, CA, USA, August 12-13, 2019*. USENIX Association, 2019.

[41] Yoongu Kim et al. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*. IEEE Computer Society, 2014, pp. 361–372.

[42] Vivek Kothari, Edgar Liberis, and Nicholas D. Lane. "The Final Frontier: Deep Learning in Space". In: *HotMobile '20: The 21st International Workshop on Mobile Computing Systems and Applications, Austin, TX, USA, March 3-4, 2020*. ACM, 2020, pp. 45–49.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. 2012, pp. 1106–1114.

[44] Yann LeCun, John S. Denker, and Sara A. Solla. "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*. Morgan Kaufmann, 1989, pp. 598–605.

[45] He Li, Qiang Liu, and Jiliang Zhang. "A survey of hardware Trojan threat and defense". In: *Integr.* 55 (2016), pp. 426–437.

[46] Wenshuo Li et al. "Hu-Fu: Hardware and Software Collaborative Attack Framework Against Neural Networks". In: *2018 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2018, Hong Kong, China, July*

8-11, 2018. IEEE Computer Society, 2018, pp. 482–487.

[47] Yuanchun Li, Jiayi Hua, Haoyu Wang, Chunyang Chen, and Yunxin Liu. "DeepPayload: Black-box Backdoor Attack on Deep Learning Models through Neural Payload Injection". In: *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 263–274.

[48] Qi Liu, Jieming Yin, Wujie Wen, Chengmo Yang, and Shi Sha. "NeuroPots: Realtime Proactive Defense against Bit-Flip Attacks in Neural Networks". In: *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023, pp. 6347–6364.

[49] Wenye Liu, Chip-Hong Chang, Fan Zhang, and Xiaoxuan Lou. "Imperceptible Misclassification Attack on Deep Learning Accelerator by Glitch Injection". In: *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. IEEE, 2020, pp. 1–6.

[50] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. "Fault injection attack on deep neural network". In: *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017*. IEEE, 2017, pp. 131–138.

[51] Yingqi Liu et al. "Trojaning Attack on Neural Networks". In: *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[52] Christos Louizos, Karen Ullrich, and Max Welling. "Bayesian Compression for Deep Learning". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 2017, pp. 3288–3298.

[53] Christos Louizos, Max Welling, and Diederik P. Kingma. "Learning Sparse Neural Networks through L_0 Regularization". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[54] Eric Love, Yier Jin, and Yiorgos Makris. "Proof-Carrying Hardware Intellectual Property: A Pathway to Trusted Module Acquisition". In: *IEEE Trans. Inf. Forensics Secur.* 7.1 (2012), pp. 25–40.

[55] Hua Ma et al. "Quantization Backdoors to Deep Learning Commercial Frameworks". In: *IEEE Trans. Dependable Secur. Comput.* 21.3 (2024), pp. 1155–1172.

[56] Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. "Variational Dropout Sparsifies Deep Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 2498–2507.

[57] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. "On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*. ACM, 2011, pp. 111–124.

[58] Amir Moradi, Markus Kasper, and Christof Paar. "Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures - An Analysis of the Xilinx Virtex-4 and Virtex-5 Bitstream Encryption Mechanism". In: *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*. Vol. 7178. Lecture Notes in Computer Science. Springer, 2012, pp. 1–18.

[59] Amir Moradi and Tobias Schneider. "Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series". In: *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*. Vol. 9689. Lecture Notes in Computer Science. Springer, 2016, pp. 71–87.

[60] Rijoy Mukherjee and Rajat Subhra Chakraborty. "Novel Hardware Trojan Attack on Activation Parameters of FPGA-Based DNN Accelerators". In: *IEEE Embed. Syst. Lett.* 14.3 (2022), pp. 131–134.

[61] Adib Nahiyan et al. "Hardware trojan detection through information flow security verification". In: *IEEE International Test Conference, ITC 2017, Fort Worth, TX, USA, October 31 - Nov. 2, 2017*. IEEE, 2017, pp. 1–10.

[62] Tuan Anh Nguyen and Anh Tuan Tran. "Input-Aware Dynamic Backdoor Attack". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 2020.

[63] Jean-Baptiste Note and Éric Rannaud. "From the bitstream to the netlist". In: *Proceedings of the ACM/SIGDA 16th International Symposium on Field Programmable Gate Arrays, FPGA 2008, Monterey, California, USA, February 24-26, 2008*. ACM, 2008, p. 264.

[64] Tolulope A. Odetola, Hawzhin Raoof Mohammed, and Syed Rafay Hasan. "A Stealthy Hardware Trojan Exploiting the Architectural Vulnerability of Deep Learning Architectures: Input Interception Attack (IIA)". In: *CoRR* abs/1911.00783 (2019). arXiv: 1911.00783.

[65] Nicolas Papernot, Patrick D. McDaniel, Arunesh Sinha, and Michael P. Wellman. "SoK: Security and Privacy in Machine Learning". In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 2018, pp. 399–414.

[66] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. "Deep Face Recognition". In: *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015*. BMVA Press, 2015, pp. 41.1–41.12.

[67] Khoa Dang Pham, Edson L. Horta, and Dirk Koch. "BITMAN: A tool and API for FPGA bitstream manipulations". In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*. IEEE, 2017, pp. 894–897.

[68] Endres Puschner et al. "Red Team vs. Blue Team: A Real-World Hardware Trojan Detection Case Study Across Four Modern CMOS Technology Generations". In: *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 56–74.

[69] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. "Bit-Flip Attack: Crushing Neural Network With Progressive Bit Search". In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 1211–1220.

[70] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. "TBT: Targeted Neural Network Attack With Bit Trojan". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp. 13195–13204.

[71] Michael Rathmair, Florian Schupfer, and Christian Krieg. "Applied formal methods for hardware Trojan detection". In: *IEEE International Symposium on Circuits and Systemss, ISCAS 2014, Melbourne, Victoria, Australia, June 1-5, 2014*. IEEE, 2014, pp. 169–172.

[72] Albert Reuther et al. "Survey and Benchmarking of Machine Learning Accelerators". In: *2019 IEEE High Performance Extreme Computing Conference, HPEC 2019, Waltham, MA, USA, September 24-26, 2019*. IEEE, 2019, pp. 1–9.

[73] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. "Hidden Trigger Backdoor Attacks". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 11957–11965.

[74] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. "Dynamic Backdoor Attacks Against Machine Learning Models". In: *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*. IEEE, 2022, pp. 703–718.

[75] Senate of the United States. *CHIPS and Science Act 2022 (P.L. 117-167)*. Washington, D.C., July 2022.

[76] Ali Shafahi et al. "Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 2018, pp. 6106–6116.

[77] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*. 2014.

[78] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[79] Sergei Skorobogatov and Christopher Woods. "Breakthrough Silicon Scanning Discovers Backdoor in Military Chip". In: *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*. Vol. 7428. Lecture Notes in Computer Science. Springer, 2012, pp. 23–40.

[80] Julian Speith et al. "How Not to Protect Your IP - An Industry-Wide Break of IEEE 1735 Implementations". In: *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 1656–1671.

[81] Suraj Srinivas, Akshayvarun Subramanya, and R. Venkatesh Babu. "Training Sparse Neural Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 455–462.

[82] Cynthia Sturton, Matthew Hicks, David A. Wagner, and Samuel T. King. "Defeating UCI: Building Stealthy and Malicious Hardware". In: *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*. IEEE Computer Society, 2011, pp. 64–77.

[83] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. "Mind Your Weight(s): A Large-scale Study on Insufficient Machine Learning Model Protection in Mobile Apps". In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. USENIX Association, 2021, pp. 1955–1972.

[84] Pawel Swierczynski, Amir Moradi, David F. Oswald, and Christof Paar. "Physical Security Evaluation of the Bitstream Encryption Mechanism of Altera Stratix II and Stratix III FPGAs". In: *ACM Trans. Reconfigurable Technol. Syst.* 7.4 (2015), 34:1–34:23.

[85] Shahin Tajik, Heiko Lohrke, Jean-Pierre Seifert, and Christian Boit. "On the Power of Optical Contactless Probing: Attacking Bitstream Encryption of FPGAs". In: *Proceedings of the 2017 ACM SIGSAC Conference*

on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. ACM, 2017, pp. 1661–1674.

[86] Di Tang, XiaoFeng Wang, Haixu Tang, and Kehuan Zhang. "Demon in the Variant: Statistical Analysis of DNNs for Robust Backdoor Contamination Detection". In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. USENIX Association, 2021, pp. 1541–1558.

[87] Ruixiang Tang, Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. "An Embarrassingly Simple Approach for Trojan Attack in Deep Neural Networks". In: *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 2020, pp. 218–228.

[88] Mohammad Tehranipoor and Farinaz Koushanfar. "A Survey of Hardware Trojan Taxonomy and Detection". In: *IEEE Des. Test Comput.* 27.1 (2010), pp. 10–25.

[89] M. Caner Tol, Saad Islam, Andrew J. Adiletta, Berk Sunar, and Ziming Zhang. "Don't Knock! Rowhammer at the Backdoor of DNN Models". In: *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Network, DSN 2023, Porto, Portugal, June 27-30, 2023*. IEEE, 2023, pp. 109–122.

[90] Brandon Tran, Jerry Li, and Aleksander Madry. "Spectral Signatures in Backdoor Attacks". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 2018, pp. 8011–8021.

[91] Karen Ullrich, Edward Meeds, and Max Welling. "Soft Weight-Sharing for Neural Network Compression". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[92] Bolun Wang et al. "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 707–723.

[93] Chao Wang et al. "DLAU: A Scalable Deep Learning Accelerator Unit on FPGA". In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 36.3 (2017), pp. 513–517.

[94] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. "HAQ: Hardware-Aware Automated Quantization With Mixed Precision". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 8612–8620.

[95] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. "Learning Structured Sparsity in Deep Neural Networks". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, pp. 2074–2082.

[96] Carole-Jean Wu et al. "Machine Learning at Facebook: Understanding Inference at the Edge". In: *25th IEEE International Symposium on High Performance Computer Architecture, HPCA 2019, Washington, DC, USA, February 16-20, 2019*. IEEE, 2019, pp. 331–344.

[97] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation". In: *CoRR* abs/2004.09602 (2020). arXiv: 2004.09602.

[98] Kan Xiao et al. "Hardware Trojans: Lessons Learned after One Decade of Research". In: *ACM Trans. Design Autom. Electr. Syst.* 22.1 (2016), 6:1–6:23.

[99] Xiaojun Xu et al. "Detecting AI Trojans Using Meta Neural Analysis". In: *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 103–120.

[100] Mingfu Xue, Chongyan Gu, Weiqiang Liu, Shichao Yu, and Máire O'Neill. "Ten years of hardware Trojans: a survey from the attacker's perspective". In: *IET Comput. Digit. Tech.* 14.6 (2020), pp. 231–246.

[101] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. "DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips". In: *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*. USENIX Association, 2020, pp. 1463–1480.

[102] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao. "Latent Backdoor Attacks on Deep Neural Networks". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. ACM, 2019, pp. 2041–2055.

[103] Jing Ye, Yu Hu, and Xiaowei Li. "Hardware Trojan in FPGA CNN Accelerator". In: *27th IEEE Asian Test Symposium, ATS 2018, Hefei, China, October 15-18, 2018*. IEEE, 2018, pp. 68–73.

[104] Haoti Zhong, Cong Liao, Anna Cinzia Squicciarini, Sencun Zhu, and David J. Miller. "Backdoor Embedding in Convolutional Neural Network Models via Invisible Perturbation". In: *CODASPY '20: Tenth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, March 16-18, 2020*. ACM, 2020, pp. 97–108.

[105] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. "Adaptive Quantization for Deep Neural Network". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. AAAI Press, 2018, pp. 4596–4604.