Resource Efficient Deep Learning Hardware Watermarks with Signature Alignment

Joseph Clements[1,2], **Yingjie Lao**[1,3]
[1]Clemson University, Clemson, South Carolina, 29634
[2]Applied Research Associates, Albuquerque, New Mexico, 87110
[3]Tufts University, Medford, Massachusetts, 02155
jclements@ara.com, yingjie.lao@tufts.edu

Abstract

Deep learning intellectual properties (IPs) are high-value assets that are frequently susceptible to theft. This vulnerability has led to significant interest in defending the field's intellectual properties from theft. Recently, watermarking techniques have been extended to protect deep learning hardware from privacy. These technique embed modifications that change the hardware's behavior when activated. In this work, we propose the first method for embedding watermarks in deep learning hardware that incorporates the owner's key samples into the embedding methodology. This improves our watermarks' reliability and efficiency in identifying the hardware over those generated using randomly selected key samples. Our experimental results demonstrate that by considering the target key samples when generating the hardware modifications, we can significantly increase the embedding success rate while targeting fewer functional blocks, decreasing the required hardware overhead needed to defend it.

Introduction

Deep learning has seen a significant rise to prominence in recent years due to various technological advancements (Dave et al. 2022; Bernstein et al. 2021; Lucas et al. 2021). Systems powered by this technology have displaced conventional approaches in many domains. Unfortunately, deep learning systems require heavy data acquisition and processing, expertise, and computing resources (Thompson et al. 2021). Further, many works have demonstrated that the security of such systems is a significant concern (Chakraborty et al. 2021; Mani, Moh, and Moh 2021; Zhao and Lao 2022; Allen-Zhu and Li 2022). These factors make deep learning Intellectual Properties (IPs) high-value assets that are prime targets for modern piracy (Wei et al. 2020; Yu et al. 2020). In addition, the theft of deep learning IPs can often be trivial such as through model stealing attacks (Tramèr et al. 2016). Watermarks create avenues of detecting stolen IPs and remedying such property violations (Zhu et al. 2021).

Due to the negative impacts of piracy, protecting deep learning IPs will become increasingly important to encourage healthy practices in the field. However, defenses for mitigating deep learning IP piracy are in their early stages and have predominately focused on defending deep learning models from theft (Zhu et al. 2021; Xue, Wang, and Liu 2021). Deep learning watermarks embed the owner's signature into a model, often through an identifiable behavior in response to specific inputs (i.e., key samples) (Adi et al. 2018). The embedding of this signature is usually achieve through the utilization of the injection of non-adversarial backdoors into the model. After embedding, the presence of the watermark should be easily verifiable by using the key samples even without direct access to the model. Thus, watermarks can allow the reclaiming of the model after a theft has occurred and are often the final defense against piracy.

Other deep learning IPs are similarly vulnerable to theft. Recent interest has been directed at defending other deep learning components, such as training data (Zou, Gong, and Wang 2021). Meanwhile, the hardware on which a deep learning model is executed has significant impact on the system's performance and is known to be vulnerable through modern supply chains. Various deep learning hardware accelerators have been developed which show significant improvement in speed and efficiency (Talib et al. 2021; Peng et al. 2021; Azghadi et al. 2020). Introducing watermarks to deep learning hardware IPs, gives hardware developers the ability to defend their hardware designs. It has been demonstrated that by purposefully introducing well-optimized modifications into a hardware design, it is possible to inject a backdoor into a deep learning model executed on the hardware (Clements and Lao 2019).

By merging the concept of deep learning watermarks and hardware backdoors, a recent work developed DeepHardMark, the first framework for protecting deep learning hardware using watermarks (Clements and Lao 2022). Despite the success of embedding watermarks into a hardware design, the impact of the key samples is overlooked, as this previous approach exclusively utilized randomly selected key samples. However, the specific key samples used strongly correlate to the defense's efficacy and the hardware's efficiency. In this work, we expand deep learning hardware watermarks by incorporating the key samples into the watermark generation process. We propose a novel methodology for aligning the watermark's key samples with hardware modifications to produce resource efficient hardware watermarks. Specifically, this work makes the following contributions in producing deep learning hardware watermarks:

- We evaluate for the first time the impact of the key samples on the resultant hardware modifications of deep learning hardware watermarks.
- We develop a hardware modification model, establishing a link between the effect of hardware modifications in the algorithmic domain to the hardware.
- We propose an algorithm that attempts to simultaneously embed an aligned signature using both the key sample and key DNN.
- We demonstrate that watermark embeddings produced using our optimal key samples provide superior characteristics to those produced with previous methods.

Related Works

Deep Learning Watermarking

Watermarks in deep learning have primarily defended the algorithmic IP from theft by embedding a signature, a unique property or functionality, into the model to identify the IP. The first methods of embedding watermarks in deep neural networks did so directly modifying model parameters (Uchida et al. 2017). Some approaches build upon this perspective (Zhang et al. 2021), but verifying such watermarks often requires analyzing the weights directly. Methods that can embed the signature into the model's response to specific inputs have become popular (Rouhani, Chen, and Koushanfar 2018; Lu 2022; Quan et al. 2020). A frequently used method for producing this type of watermark applies backdoor injection algorithms to alter the model (Adi et al. 2018; Yang et al. 2021; Zhang et al. 2018).

Methods for improving the characteristics of deep learning watermarks, such as capacity and the robustness of the watermarks, have become a significant research focus (Zhu et al. 2021; Lv et al. 2021; Ye et al. 2022). Recently, many works have extended the use of deep learning watermarks to various high-profile deep learning scenarios such as deep reinforcement learning (Chen et al. 2021), federated learning (Li, Wang, and Liew 2022), and multi-task learning (Li and Wang 2021). Work from the adversarial perspective has drawn concerns that ambiguity attacks can compromise deep learning watermarks (Fan, Ng, and Chan 2019). Currently, very few methods can defend against such attacks (Ong et al. 2021). Currently, similar vulnerabilities have not been observed from the hardware perspective and known methods cannot be trivially extended due to the discrete nature of hardware modifications limiting gradient descent-based methods (Clements and Lao 2022).

Deep Learning Hardware Watermarking

To the best of our knowledge, DeepHardMark is the only prior work to apply backdoor inspired watermarking towards defending deep learning hardware IPs from piracy (Clements and Lao 2022). Other works have proposed the integration of hardware platforms into the protection of deep learning models by degrading its performance on non-approved devices (Goldstein et al. 2021), which are very different from watermarking. Figure 1 depicts a flowchart describing a general scheme for embedding and verifying hardware-based watermarks in deep learning systems. Instead of embedding the watermark functionality into weights, the DeepHardMark provides a scheme for embedding the signature through modifications in its hardware. Consequently, key DNN and key samples are needed to verify the hardware watermark instead of requiring only the key samples, as with algorithmic watermarks.

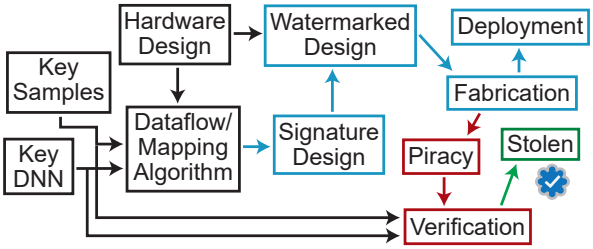

Figure 1: A general scheme for deep learning hardware watermarks embeds a signature linked to a specific DNN and input samples during the hardware design process. The owner can utilize the key DNN and samples to verify ownership over a pirated device.

There are many distinctions between algorithmic and hardware watermarks due to the differences in the two domains. For example, the immutability of hardware after fabrication provides the benefit of hardware watermarks being more naturally robust to removal attacks. Deep learning hardware watermarks also come with the additional requirement of minimizing the overhead of the watermark. Finally, it is also necessary to preserve the original algorithmic response of the system over a broad range of models since hardware platforms are usually expected to accommodate the rapid development in this field with various models.

Limitations of Prior Method

Of critical importance in protecting deep learning hardware IPs with watermarks is the impact in terms of the hardware overhead and the device's functional behavior. A rudimental understanding of hardware indicates that the novel hardware used to implement the modifications would introduce additional area, power, and propagation delay to the circuit. As such, reducing the number of modifications directly correlates with a reduction in the hardware overhead. Prior works have also concluded that there is a connection between the number of hardware modifications and the device's functional behavior (Clements and Lao 2022) due to a small probability of the modifications being erroneously activated. While deep learning systems tend to be reasonably robust to such noises, as the number of modifications increase the possibility of unexpected alterations to the algorithmic behavior of DNNs executed on the device also increases.

As such, a focus of this prior work was to reduce the number of modifications injected into the target hardware using a cardinality constraint (ϵ_η). This constraint enforces an upper bound on the number of model operations targeted by the watermark, which is a major component in determining the magnitude of hardware modifications embedded. Here, we

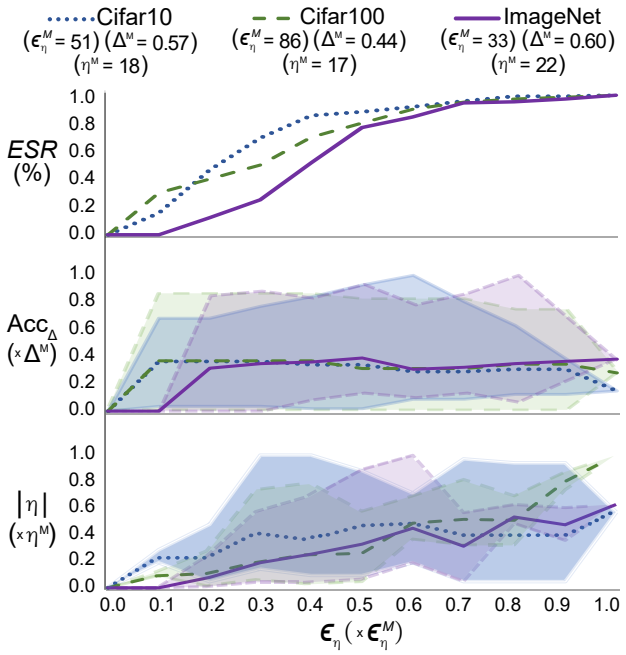

Figure 2: Prior works attempt to control the impact of the watermark through a cardinality constraint (ϵ_η). We observe a greater deviation based on the key samples and propose to align the key sample with the signature.

provide a preliminary experimental evaluation of this perspective and demonstrate its limitations.

We perform this experimental evaluation using the ResNet18 (He et al. 2016) architecture trained for Cifar10, Cifar100 (Krizhevsky, Hinton et al. 2009), and ImageNet (Deng et al. 2009) as the key DNN. We perform a baseline hardware watermark embedding for 100 randomly selected key samples while iteratively increasing ϵ_η until a successful watermark embedding is found for each key sample. We categorize each test case based on the value of ϵ_η used in generating them. Then, we evaluate these groups using the Embedding Success Rate (ESR), Change in Accuracy (Acc_Δ), and the number of targeted modifications ($|\eta|$), which are consistent with the metrics used in DeepHardMark (Clements and Lao 2022). Formal definitions are presented in Section , but intuitively speaking, they correlate with the ease of watermark embedding, algorithmic impact, and hardware impact, respectively. The results of this evaluation are shown in Figure 2, which are normalized with respect to the maximum value for each test.

In this evaluation, we observe that while there is a general correlation between ESR, or $|\eta|$, and ϵ_η, the same does not appear to be true for Acc_Δ. This observation implies that the reduction of the cardinality constraint does little to preserve the algorithmic behavior of the key DNN. We further observe a wide variation in these metrics at all values of ϵ_η. This observation leads to the fact that, in these tests, we can often increase the watermark's performance with respect to these metrics by up to 70% simply by selecting an average performing key sample from the original randomly set rather than a bad one, even at the same ϵ_η.

These results demonstrate that the key samples used in generating the hardware watermark contribute significantly to its overall performance. From this perspective, we propose that by utilizing a methodology that aligns the key samples with the watermark signature, generating a much more efficient watermark embedding is possible. We propose a novel watermark embedding algorithm to accomplish this.

Signature Aligned Hardware Watermarks

Threat Model IP piracy is a significant concern of hardware security in the current semiconductor industry. As a modern hardware device typically involves several design houses, foundries, and 3rd-party IPs that span multiple countries, such a supply chain readily allows adversaries access to low-level hardware designs outside the view of the developer. The adversary's goal is to steal the IP and profit from counterfeit products, e.g., sell the counterfeit products to customers, which would subvert the owner's profit.

We consider a scenario where an adversary has acquired and deployed a pirated deep learning hardware IP. To maximize utilization of the device the adversary deploys the device in a setting where users provide the DNN, $F(\cdot)$, and inputs, $\mathbf{x}$, to be processed. In this scenario, the designer's objective is to embed a watermark into the device through modifications to its circuitry. To confirm the watermark's presence, the owner would characterize the hardware on open source deep learning systems, then demonstrate a deviation from the expected behavior when processing the owner's key DNN and key sample. From a high-level perspective this scenario appears similar to prior methodologies seen in the software domain. It should be noted that intuitions from the software domain do not necessarily generalize to the hardware. For example, modifications in the hardware do not necessarily effect model gradients in the same way perturbations on the weights do.

Embedding Hardware Watermarks

The typical algorithmic perspective of deep learning considers a DNN, $F(\cdot)$, to be a function trained to map inputs, $\mathbf{x}$, to corresponding model responses, $\mathbf{y}$. The model's ability to produce this mapping is the algorithmic response of the model. This algorithmic response is made to match a dataset, $\mathcal{D} = \{\mathbf{x}, \mathbf{y}\}$, representing this task according to a loss function, $\mathcal{L}$, which quantifies the error over the task. Due to the scale of modern DNNs and physical limitations on hardware resources, during the execution of a DNN, model operations are mapped onto functional blocks, which multiple operations must share in a time-multiplexed manner. Various optimization techniques have been incorporated to accelerate the execution, including multiple dataflow schemes which specify how operations are mapped into the hardware (Sze et al. 2017). We understand the ability of the hardware design to produce this algorithmic response correctly as the hardware's functional behavior. Modifications to the hardware often alter the functional behavior of the design. If these modifications are well-aligned with a DNN, $F(\cdot)$, they can even produce targeted changes in the algorithmic response (Clements and Lao 2018).

The central objective of the process is to alter the algorithmic functionality of a *key DNN*, $F_k(\cdot)$, on specific *key samples*, $\mathbf{x}_k$, when executed on the watermarked hardware. We distinguish the model executed on the altered hardware as $F^{\delta}(\cdot)$, signifying the change in its algorithmic response. Formally, the altered functionality is described as: $F_k^{\delta}(\mathbf{x}_k) = \hat{\mathbf{y}}_k$, where $\hat{\mathbf{y}}_k$ sharply deviates from the expected algorithmic response indicated by $\mathcal{D}$. The watermark embedding should produce this deviation while preserving its algorithmic functionality in other scenarios.

In addition to this central objective, we want to reduce the impact of the watermark on the hardware. Following the observation discussed in Section that both the selected key sample and magnitude of the hardware modification play a significant role in this endeavor, we propose an algorithm to target both aspects. We establish a link between the hardware modifications and the algorithmic domain to accomplish this. While the effect of the modifications on the algorithmic response is not equivalent to direct perturbations on the DNN, we can approximate it layer-wise perturbation, $\delta = \{\delta_l\}$. For a targeted operation, op, in layer, l, with parameters, W_l, of a DNN $H_l = op(W_l, H_{l-1})$, the perturbed latent representation is: $\widehat{H}_l = op(W_l, H_{l-1}) + \delta_l$. Further, we also extend the understanding of the key sample selection as perturbing a known input $\hat{\mathbf{x}}_k = \mathbf{x}_k + \mathbf{p}$. Finding a selection for $\mathbf{p}$ and δ, which align with the watermark signature, allows us to minimize both the hardware overhead and algorithmic functionality simultaneously.

Modeling Hardware Modifications

To begin, we assume the owner of the device has access to a model to be used as the key DNN, $F_k(\cdot)$, trained on a dataset, $\mathcal{D}$. The owner can then randomly select a subset of samples from the training dataset with altered labels signifying the watermark signature, i.e., $\mathcal{D}_k = \{\mathbf{x}_k, \hat{\mathbf{y}}_k\}$, such that $\hat{\mathbf{y}}_k \neq \mathbf{y}_k$. In prior algorithms embedding the watermark can be understood as simply establishing a link between hardware and the key samples, $\mathbf{x}_k$. We extend this perspective by simultaneously co-optimizing the sample input to find a new sample, $\hat{\mathbf{x}}_k \approx \mathbf{x}_k$. This sample should meet the criteria that $F_k(\mathbf{x}_k) = F_k(\hat{\mathbf{x}}_k) = \mathbf{y}_k$ and $F_k^{\delta}(\mathbf{x}_k) \neq F_k^{\delta}(\hat{\mathbf{x}}_k) = \hat{\mathbf{y}}_k$ while minimizing the necessary hardware modifications.

The link is established through hardware modifications, which alter the functionality of the key DNN producing the result $F_k^{\delta}(\hat{\mathbf{x}}_k) = \hat{\mathbf{y}}_k$. We identify a class of hardware modifications that can produce the perturbations, δ_l, on the latent representations, H_l, as they are being computed in hardware. Research into hardware Trojan attacks has produced various avenues for designing this class of modification while having a minimal impact on both the functionality and hardware overhead (Chakraborty, Mondal, and Srivastava 2020). One of the most straightforward methods is to utilize two small logic circuits, which are referred to as the *activator* ($act(\cdot,\cdot)$) and *perturber* ($per(\cdot,\cdot,\cdot)$). When embedded in a functional block, $act(\cdot,\cdot)$ is designed to detect when a target operation is being processed, which sends a signal to $per(\cdot,\cdot,\cdot)$. Then, $per(\cdot,\cdot,\cdot)$ responds to this signal by altering the operation of the functional blocks in a way that emulates a perturbation on the latent space. Figure 3 presents a block diagram depicting this modification scheme.

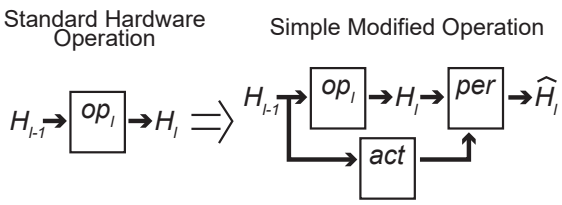

Figure 3: An activator (act) and perturbator (per) circuit can be introduced to the hardware to modify its outputs.

We mathematically define the activators as $t_{l,op_t} = act(H_{l-1,op_t}, T_{l-1,op_t})$ where l and op_t signify the layer and specific target operation in the layer which should activate $act(\cdot,\cdot)$. The functionality of this circuit is to analyze the incoming latent representations, H_{l-1}, during computation and produce a binary response signal t_{l,op_t} signifying that the signal was detected. Our approach is to make a static comparison that evaluates and compares the binary latent representation observed, $[H_{l-1}]_{bin}$, with the binary latent representation expected for the key sample, $[T_{l-1}]_{bin}$.

$$t_{l,op_t} = \mathbb{1}[comp([H_{l-1,op_t}]_{bin}, [T_{l-1,op_t}]_{bin})] \quad (1)$$

where $comp$ is a binary comparison and $\mathbb{1}[cond]$ is the indicator function that returns 1 when $cond$ is a True statement and 0 if False. t_{l,op_t} is a trigger signal sent to the perturber indicating that it should produce the required perturbation.

If activated, the perturber should alter the computation of the block to produce the desired perturbation. We want $per(\cdot,\cdot,\cdot)$ to perform the operation $\widehat{H}_l = H_l + \delta_l$ given that we know $H_l = T_l$, i.e. the latent representation for the target operation. Performing this in hardware can be done with the following mathematical representation of $per(t,h,b)$ where $t = t_l$, $h = H_l$, and $b = \beta_l$ for a given l.

$$per(t,h,b) = \begin{cases} [h_{op_t}]_{bin} \oplus [b_{op_t}]_{bin}, & \text{when } t_{op_t} = 1, \\ [h_{op_t}]_{bin}, & \text{otherwise.} \end{cases} \quad (2)$$

where $\oplus$ is the *XOR* operation used to flip the bits of $[h_{op_t}]_{bin}$ based on the corresponding binary mask $[b_{op_t}]_{bin} \in [\beta_{l,op_t}]_{bin}$. $[\beta_{l,op_t}]_{bin}$ should be selected such that $[T_{l,op_t}]_{bin} \oplus [\beta_{l,op_t}]_{bin} = [T_{l,op_t} + \delta_{l,op_t}]_{bin}$ implying that $[\beta_{l,op_t}]_{bin} = [T_{l,op_t}]_{bin} \oplus [T_{l,op_t} + \delta_{l,op_t}]_{bin}$.

In summary, the hardware designer needs three parameters T_l, T_{l-1}, and δ_l. T_l and T_{l-1} can determined by probing the model and observing its latent representations when computing $\hat{\mathbf{x}}_k$. The functionality of $F_k^{\delta}(\hat{\mathbf{x}}_k)$ can be simulated. Then, using modern optimization techniques the δ_l required to produce their watermark signature is determined. These circuits consume some hardware overhead and have the potential of activating under untargeted input samples. The designer should reduce the impact of the modifications on the hardware by minimize its the magnitude of δ_l.

Perturbation/Sample Signature Alignment

From the algorithmic perspective, we wish to determine a δ which produces the watermark embedding defined by

$F_k^{\delta}(\hat{\mathbf{x}}_k) = \hat{\mathbf{y}}_k \neq F_k(\mathbf{x}_k)$. Our approach to solve for δ is with the optimization problem:

$$\underset{\delta}{\text{minimize}} \quad \mathcal{L}(F_k^{\delta}(\hat{\mathbf{x}}_k), \hat{\mathbf{y}}_k). \tag{3}$$

where $\mathcal{L}$ is a standard deep learning loss function for the corresponding task. For classifiers, we utilize cross-entropy loss and an arbitrarily selected abnormal label, $\hat{\mathbf{y}}_k$. However, an essential criterion of a watermark is to minimize any impacts on the design. One method of lowering the hardware overhead is to decrease the number of modifications. This may also help reduce incorrect activation of the modifications due to there being less modifications to activate. Using the combinational circuits to produce the perturbation as describe above, each non-zero element of δ requires additional hardware. Thus, a significant component of the watermarking strategy is enforcing sparsity in δ.

To meet this objective, we utilize a binary mask, $\eta = \{\eta_l\}$ for each layer l, which selects which operations in $F_k(\cdot)$ are targeted for modification. Using this mask, we compose a block constrained perturbation $\widehat{\delta} = \delta \odot \eta$. By setting ensuring the sparsity of η, we can ensure the sparsity of $\widehat{\delta}$. Utilizing this as our model perturbation, we reformulate the problem as follows:

$$\begin{aligned} \underset{\delta,\eta}{\text{minimize}} \quad & \mathcal{L}(F_k^{\delta\odot\eta}(\hat{\mathbf{x}}_k), \hat{\mathbf{y}}_k) \\ \text{subject to} \quad & |\eta| < \epsilon_\eta,\ \eta_i \in \{0,1\} \end{aligned} \tag{4}$$

Here ϵ_η becomes a constraint that determines the upper bound on the number of operations targeted for modification. Solving this problem ensures that the number of modification circuits needed to be embedded is less than ϵ_η.

Samples near a classification boundary require a smaller model perturbation to cross the decision boundary. We can select an optimal $\hat{\mathbf{x}}_k$ by beginning with a seed image selected from the training dataset and introducing a small perturbation to it, i.e., $\hat{\mathbf{x}}_k = \mathbf{x}_k + \mathbf{p}$. We can then optimize $\mathbf{p}$ to minimize $\mathcal{L}(F_k^{\delta\odot\eta}(\mathbf{x}_k + \mathbf{p}), \hat{\mathbf{y}}_k)$ while solving Equation 4. However, the best solution for this problem occurs when $\eta = 0$, which can be seen as an adversarial example for the unmodified key DNN, $F_k(\mathbf{x}_k + \mathbf{p}) = \hat{\mathbf{y}}_k$. As such, for the key sample to be useful in demonstrating the functional difference between the watermarked and clean hardware designs, we must ensure that it does not collapse into an adversarial example for the key DNN. As such, we must introduce the constraint that $F_k(\mathbf{x}_k + \mathbf{p}) = \mathbf{y}_k$, the original label. Finally, we can formulate the full optimization problem as follows:

$$\begin{aligned} \underset{\delta,\eta,\mathbf{p}}{\text{minimize}} \quad & \mathcal{L}(F_k^{\delta\odot\eta}(\mathbf{x}_k + \mathbf{p}), \hat{\mathbf{y}}_k) \\ \text{subject to} \quad & |\eta| < \epsilon_\eta,\ \eta_i \in \{0,1\}, \\ & F_k(\mathbf{x}_k + \mathbf{p}) = \mathbf{y}_k,\ \|\mathbf{p}\|_\infty < \epsilon_p \end{aligned} \tag{5}$$

We introduce the additional ϵ_p constraint to ensure that the key sample retains some similarity to $\mathbf{x}_k$.

Algorithm 1: Signature Alignment Algorithm

Require: $\mathcal{L}$, F_k, $(\mathbf{x}_k, \mathbf{y}_k)$, $\hat{\mathbf{y}}_k$, $\tau_{\{1,2,3\}}$, c_p $\alpha_{\{\delta,p,\eta\}}$, ϵ_L
$\boldsymbol{\zeta}_1 = \boldsymbol{\zeta}_2 = \boldsymbol{\zeta}_3 = \mathbf{0}$; $\epsilon_\eta = \epsilon_L[0]$
while Inner Loops Converge **do**
  while $|\boldsymbol{\eta}| >= \epsilon_\eta$ & **not** timed_out **do**
    $L = \mathcal{L}(F_k^{\boldsymbol{\delta}\odot\boldsymbol{\eta}}(\mathbf{x}_k + \mathbf{p}), \hat{\mathbf{y}}_k)$
    $A_1 = \tau_1(\boldsymbol{\eta} - \mathbf{S}_1) + \boldsymbol{\zeta}_1$
    $A_2 = \tau_2(\boldsymbol{\eta} - \mathbf{S}_2) + \boldsymbol{\zeta}_2$
    $A_3 = \tau_3(\mathbf{1}^T\boldsymbol{\eta} - \epsilon_\eta) + \boldsymbol{\zeta}_3$
    $\boldsymbol{\eta} = \boldsymbol{\eta} - \alpha_\eta[\frac{\partial L}{\partial \boldsymbol{\eta}} + A_1 + A_2 + A_3]$
    $\boldsymbol{\zeta}_i \mathrel{+}= \tau_i(\boldsymbol{\eta} - \boldsymbol{\zeta}_i)\ \forall i \in \{1,2\};\ \boldsymbol{\zeta}_3 = A_3$
  end while
  while $F_k^{\boldsymbol{\delta}\odot\boldsymbol{\eta}}(\mathbf{x}_k + \mathbf{p}) \neq \hat{\mathbf{y}}_k$ & **not** timed_out **do**
    $L_1 = \mathcal{L}(F_k^{\boldsymbol{\delta}\odot\boldsymbol{\eta}}(\mathbf{x}_k + \mathbf{p}), \hat{\mathbf{y}}_k)$
    $L_2 = \mathcal{L}(F_k(\mathbf{x}_k + \mathbf{p}), \mathbf{y}_k)$
    $\mathbf{p} = \mathbf{p} - \alpha_p[\frac{\partial L_1}{\partial \mathbf{p}} + c_p\frac{\partial L_2}{\partial \mathbf{p}}]$
    $\boldsymbol{\delta} = \boldsymbol{\delta} - \alpha_\delta\frac{\partial L_1}{\partial \boldsymbol{\delta}}$
  end while
end while
$\epsilon_\eta = \text{next}(\epsilon_L)$

Optimization Strategy

Due to the binary mask representation of η, we recognize that optimizing η is akin to minimizing the so-called ℓ_0-norm seen deep learning application domains such as sparse adversarial examples. A strong solution to such problems is notoriously difficult to find due to the discrete nature of the variable. Recently, a novel algorithm was proposed for solving similarly mixed integer problems (Wu and Ghanem 2019). Using this work as inspiration, we are able to develop a process for solving the optimization problem laid out in Equation 5.

To solve this problem, we employ an algorithm that cycles through multiple iterations of updating η followed by iteratively updating δ and $\mathbf{p}$ simultaneously. The full algorithm is presented in Algorithm 1. We begin by first updating δ as follows:

$$\delta = \delta - \alpha_\delta \left[\frac{\partial \mathcal{L}(F_k^{\delta\odot\eta}(\mathbf{x}_k + \mathbf{p}), \hat{\mathbf{y}}_k)}{\partial \delta}\right] \tag{6}$$

where α_δ is the step size that controls the convergence rate for δ. Simultaneously, we update $\mathbf{p}$ using:

$$\mathbf{p} = \mathbf{p} - \alpha_p\, d\mathbf{p} \tag{7}$$

where

$$d\mathbf{p} = \frac{\partial \mathcal{L}(F_k^{\delta\odot\eta}(\mathbf{x}_k + \mathbf{p}), \hat{\mathbf{y}}_k) + c_P\, \mathcal{L}(F_k(\mathbf{x}_k + \mathbf{p}), \mathbf{y}_k)}{\partial \mathbf{p}} \tag{8}$$

Here, α_p is the step size, and c_P is a constant used to balance the strength of the terms. These parameters control the convergence rate and the degree to which the solution avoids the trivial adversarial example solution for $\hat{\mathbf{x}}_k$. Once updated, we project $\mathbf{p}$ back into the allowable solution space using $\mathbf{p}_i = min(max(\mathbf{p}_i, -\epsilon), \epsilon)$ to all $\mathbf{p}_i \in \mathbf{p}$.

Once a solution to the watermarking problem is found, we follow the ℓ_p-ADMM algorithm for reducing η (Wu and Ghanem 2019). We leave the details of this algorithm to the initial work of DeepHardMark (Clements and Lao 2022). Using this algorithm, we find that by iteratively updating η using the equation below for optimizing η:

$$\eta = \eta + \alpha_\eta \left[\frac{\partial \mathcal{L}^A}{\partial \eta} \right] \tag{9}$$

where α_η is the step size and $\frac{\partial \mathcal{L}}{\partial \eta}$ is the derivative of the augmented Lagrangian function:

$$\begin{aligned} \frac{\partial \mathcal{L}^A}{\partial \eta} =& \frac{\partial \mathcal{L}(F_k^{\delta \odot \eta}(\mathbf{x}_k + \mathbf{p}), \hat{\mathbf{y}}_k)}{\partial \eta} + \tau_1(\eta - \mathbf{S}_1) + \zeta_1 \\ &+ \tau_2(\eta - \mathbf{S}_2) + \zeta_2 + \tau_3(\mathbf{1}^T \eta - \epsilon_\eta) + \zeta_3. \end{aligned} \tag{10}$$

where, $\zeta_1 \in \mathbb{R}^M$, $\zeta_2 \in \mathbb{R}^M$, and $\zeta_3 \in \mathbb{R}^1$ are dual variables with corresponding penalty parameters: τ_1, τ_2, and τ_3. $\mathbf{S}_1$ and $\mathbf{S}_2$ are the projections of η on to an ℓ_∞-box and ℓ_2-sphere, respectively. $\mathbf{S}_1 = max(min(\mathcal{P}_{\mathcal{S}_p}(\eta + \frac{1}{\tau_1}\zeta_1), \mathbf{1}), \mathbf{0})$ and $\mathbf{S}_2 = \frac{\sqrt{M}}{2} \frac{(\eta + \frac{1}{\tau_2}\zeta_2) - 0.5(\mathbf{1})}{\|(\eta + \frac{1}{\tau_2}\zeta_2) - 0.5(\mathbf{1})\|} + \frac{1}{2}(\mathbf{1})$.

This methodology utilizes multiple step sizes and penalty parameters. The purpose of these parameters is to ensure that each of the watermark variables ($\mathbf{p}$, δ, and η) all converge at relatively the same rate to ensure the quality of the embedding. However, we note that as long as these terms are all fairly balanced, the methodology appears to generalize well from scenario to scenario and doesn't require fine-tuning for individual key samples or key DNNs. In the supplementary materials, we provide an ablation study verifying the stability of the methodology with respect to these parameters.

By iteratively performing these steps, we can find $\delta \odot \eta$, which produces the watermark functionality. η can then be used to specify which functional blocks need to be modified in the hardware, and δ used to determine $[\beta_{l,o}]_{bin}$ needed to generate $per(\cdot, \cdot, \cdot)$. This provides all the necessary information to embed watermark functionality into the hardware with the $act(\cdot, \cdot)$ and $per(\cdot, \cdot, \cdot)$ circuits. Simultaneously, we co-optimize a key sample, $\hat{\mathbf{x}}_k = \mathbf{x}_k + \mathbf{p}$, to most effectively work with the modifications to highlight the difference between the watermarked and watermark-free hardware.

Experimental Results

Experimental Setting

To demonstrate the effectiveness of the proposed methodology, we provide a comprehensive evaluation of our methodology when embedding watermarks into hardware using various well-known image classification benchmark architectures as the key DNN. Further, we demonstrate the efficacy of our algorithm by providing a comparison with DeepHardMark (Clements and Lao 2022), which to the best of our knowledge, is the only prior hardware watermark embedding framework. In this evaluation, we include ResNet (He et al. 2016), VGG (Simonyan and Zisserman 2015), and ViT (Dosovitskiy et al. 2021) models trained for the Cifar10, Cifar100, and ImageNet datasets. Finally, we present a direct evaluation of the key sample's effect on the watermark embeddings. For our experiments, we assume a hardware design composed of a grid of 128×128 ALUs with the stationary output dataflow consistent with prior hardware implementations (Putra et al. 2021). Our experiments are conducted using the Pytorch framework (Paszke et al. 2019) and the TIMM model library (Wightman 2019).

Model	$ESR\%$	$\lvert\boldsymbol{\eta}\rvert\%$	$T_{err}\%$
ResNet34	95	3.12e−3	6.15e−3
ResNet50	90	4.87e−3	2.47e−5
VGG11	100	3.27e−3	2.17e−5
VGG13	92	3.53e−3	1.52e−4
VGG16	87	2.49e−3	2.17e−5
DenseNet101	87	3.28e−3	1.82e−2
ViT-16-224	100	3.75e−2	0.00e0
ViT-32-224	100	1.65e−1	0.00e0

Table 1: Watermarks generated for ImageNet.

In these evaluations, we utilize the Embedding Success rate (ESR), Change in Accuracy (Acc_Δ), and Change in Fidelity (Fid_Δ) as proposed in prior works (Clements and Lao 2022). ESR defines the success rate when finding an embedding for specific hardware using a selected key DNN and key samples. While Acc_Δ and Fid_Δ define the change in natural prediction accuracy of the key DNN and other DNNs running on the modified hardware, respectively. We also evaluate the Triggering Error (T_{err}) by measuring the percentage of operations that produce the conditions for triggering an activator circuit (act), incorrectly. These erroneous triggerings inevitably alter model predictions and additional consumer power in the circuit, making it a strong metric for a more nuanced evaluation of the watermark's impact.

Broad Feasibility Evaluations

To demonstrate the feasibility of our methodology for embedding watermarks in deep learning hardware, we evaluate the watermark embeddings generated by our approach for various ImageNet classifiers, as shown in Table 1. In this experiment, we produce 100 watermark embedding for each model using different key samples randomly selected from the model's training data and determine the ESR for these cases. Then, using the embeddings generated, we determine the hardware modifications needed to be embedded in the hardware to produce the watermark. We then calculate $|\eta|$, T_{err}, Acc_Δ, and Fid_Δ using software simulations that emulate the functionality of the hardware modified according to Section and compare the predictions for the watermark-free model. We perform this evaluation for 5000 images randomly selected from the ImageNet Testing data. For this evaluation, we utilize a ResNet18 ImageNet classifier to calculate Fid_Δ, transferring the modification from the key DNN to the ResNet18 model according to the hardware mapping scheme. We note that across these experiments, Acc_Δ and Fid_Δ are 0%, i.e., typical DNNs running on the hardware are unchanged.

Method	Dataset	$ESR\%$	$\lvert\boldsymbol{\eta}\rvert\%$	$Acc_{\Delta}\%$	$Fid_{\Delta}\%$	$T_{err}\%$ ⋆
DeepHardMark	Cifar10	100	1.80e−1	0.68	0.12	-
	Cifar100	100	1.29e0	0.30	0.25	-
	ImageNet	100	1.50e−1	0.67	0.68	-
Signature Aligned (Proposed)	Cifar10	100	7.16e−3	0.00	0.00	0.00
	Cifar100	100	1.05e−2	0.00	0.00	0.00
	ImageNet	100	5.95e−5	0.00	0.00	1.67e−3

⋆ The values of $T_{err}\%$ are not reported in DeepHardMark (Clements and Lao 2022).

Table 2: Comparison of the proposed methodology with DeepHardMark.

Comparison with DeepHardMark

From these results, we see that for all models, the proposed methodology achieves an average ESR of 92.6%. It is possible to increase the average ESR by allowing more hardware blocks to be modified. This is done by relaxing the bound on ϵ_η until a valid solution is found. However, this leads to selecting a poor watermark embedding. Instead, we start with a large upper bound of ϵ_η and then decrease this constraint every five iterations of the algorithm by about 50% until a valid solution cannot be found. This allows us to produce more effective embeddings while sacrificing some ESR. However, this process filters out some of the suboptimal key samples contributing to a more efficient embedding. We also observe that $|\eta|$ and T_{err} are also near 0%, demonstrating the minimal impact of the hardware modifications proposed would be. These strong results would sufficiently confirm that the watermark-free and watermarked hardware are highly similar.

We also provide a direct comparison with watermarks produced by the DeepHardMark algorithm. To offer a strong comparison, we target a hardware architecture composed of a smaller ALUs grid of size 32×32 consistent with the prior work and ResNet18 key DNNs trained for the cifar10, cifar100, and ImageNet datasets. Further, the original work ensured a 100% ESR, which we also allow for this evaluation. We present the results in Table 2. From these results, we observe that our proposed methodology demonstrates superior results across all the metrics evaluated. We first note that by incorporating the key samples into the algorithm, we can observe a marked decrease in $|\eta|$, often able to beat this prior work by an order of magnitude or more. Further, of particular interest is the fact that the proposed methodology has no impact on the predictions of any of the models tested here. This is a significant boon for the proposed method, strongly demonstrating its ability to preserve the functional behavior of the device and validating the importance of key sample selection for watermarking scenarios. We further highlight that T_{err} for all scenarios in this evaluation are near zero, indicating the negligible impact of the modifications on the hardware.

Hardware Analysis

To demonstrate the efficiency of the hardware produced by the proposed approach, we utilize the TinyTPU (Shinn 2019) hardware architecture and a custom MMU architecture consistent with prior works (Clements and Lao 2022). We generate a 32×32 ASIC implementation of the design using Synopsis Design Compiler with 32nm Technology. For these implementations, we determine the area, number of cells, power utilization, and propagation delay of the design. These characteristics are presented in Table 3 alongside the results shown in DeepHardMark (Clements and Lao 2022).

Hardware Design	Area	Cells	Power	Time
	Proposed			
TinyTPU	0.001%	0.003%	0.00%	0.01%
MMU	0.034%	0.045%	0.00%	0.00%
	DeepHardMark			
TinyTPU	0.144%	0.119%	0.169%	0.00%
MMU	0.054%	0.058%	0.039%	0.00%

Table 3: Hardware Overhead of the propose methodology in ASIC designs.

From these results, we observe that the proposed methodology is highly efficient in terms of hardware overhead. The resulting watermarks produce almost no change in power and timing while reducing the number of cells and area used by the watermark by almost $100x$ in the case of TinyTPU.

Conclusions

The work proposes a novel watermark embedding algorithm for defending deep learning hardware platforms. DeepHardMark (Clements and Lao 2022), the first such methodology, did not consider the implications of integrating the key samples into the framework. This prior work attempted to minimize the watermark's impact on the design with a bound on the number of total hardware modifications. However, we observe limitations in this methodology to consistently provide optimal watermarks. To improve on this, we present a model for incorporating the hardware modifications and key samples into the watermark embedding algorithm. This aligns the key samples used to activate the watermark with its algorithmic perturbation. Hardware modifications work with the key samples to alter the behavior of a device to provide reliably verification of ownership, while having a minimal impact on the hardware.

References

Adi, Y.; et al. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *Security Symposium*, 1615–1631. USENIX Association.

Allen-Zhu, Z.; and Li, Y. 2022. Feature purification: How adversarial training performs robust deep learning. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, 977–988. IEEE.

Azghadi, M. R.; Lammie, C.; Eshraghian, J. K.; Payvand, M.; Donati, E.; Linares-Barranco, B.; and Indiveri, G. 2020. Hardware implementation of deep network accelerators towards healthcare and biomedical applications. *IEEE Transactions on Biomedical Circuits and Systems*, 14(6): 1138–1159.

Bernstein, L.; Sludds, A.; Hamerly, R.; Sze, V.; Emer, J.; and Englund, D. 2021. Freely scalable and reconfigurable optical hardware for deep learning. *Scientific reports*, 11(1): 1–12.

Chakraborty, A.; Alam, M.; Dey, V.; Chattopadhyay, A.; and Mukhopadhyay, D. 2021. A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology*, 6(1): 25–45.

Chakraborty, A.; Mondal, A.; and Srivastava, A. 2020. Hardware-Assisted Intellectual Property Protection of Deep Learning Models. In *57th Design Automation Conference (DAC), San Francisco, CA, USA, July 20-24*, 1–6. ACM/IEEE.

Chen, K.; Guo, S.; Zhang, T.; Li, S.; and Liu, Y. 2021. Temporal watermarks for deep reinforcement learning models. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 314–322.

Clements, J.; and Lao, Y. 2018. Backdoor attacks on neural network operations. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 1154–1158. IEEE.

Clements, J.; and Lao, Y. 2019. Hardware Trojan Design on Neural Networks. In *International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, May 26-29*, 1–5. IEEE.

Clements, J.; and Lao, Y. 2022. DeepHardMark: Towards Watermarking Neural Network Hardware. *Association for the Advancement of Artificial Intelligence*.

Dave, I.; Gupta, R.; Rizve, M. N.; and Shah, M. 2022. Tclr: Temporal contrastive learning for video representation. *Computer Vision and Image Understanding*, 219: 103406.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Fan, L.; Ng, K.; and Chan, C. S. 2019. Rethinking Deep Neural Network Ownership Verification: Embedding Passports to Defeat Ambiguity Attacks. In *32nd Annual Conference on Neural Information Processing Systems (NeurIPS), December 8-14, Vancouver, BC, Canada*, 4716–4725.

Goldstein, B. F.; Patil, V. C.; Ferreira, V. C.; Nery, A. S.; Franca, F. M.; and Kundu, S. 2021. Preventing DNN model IP theft via hardware obfuscation. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(2): 267–277.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 770–778. IEEE Computer Society.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Li, F.; and Wang, S. 2021. Secure Watermark for Deep Neural Networks with Multi-task Learning. *CoRR*, abs/2103.10021.

Li, F.-Q.; Wang, S.-L.; and Liew, A. W.-C. 2022. Watermarking Protocol for Deep Neural Network Ownership Regulation in Federated Learning. In *2022 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, 1–4. IEEE.

Lu, C.-S. 2022. Sparse Trigger Pattern Guided Deep Learning Model Watermarking. In *Proceedings of the 2022 ACM Workshop on Information Hiding and Multimedia Security*, 33–38.

Lucas, A. M.; Ryder, P. V.; Li, B.; Cimini, B. A.; Eliceiri, K. W.; and Carpenter, A. E. 2021. Open-source deep-learning software for bioimage segmentation. *Molecular Biology of the Cell*, 32(9): 823–829.

Lv, P.; Li, P.; Zhang, S.; Chen, K.; Liang, R.; Zhao, Y.; and Li, Y. 2021. HufuNet: Embedding the Left Piece as Watermark and Keeping the Right Piece for Ownership Verification in Deep Neural Networks. *CoRR*, abs/2103.13628.

Mani, N.; Moh, M.; and Moh, T.-S. 2021. Defending deep learning models against adversarial attacks. *International Journal of Software Science and Computational Intelligence (IJSSCI)*, 13(1): 72–89.

Ong, D. S.; Chan, C. S.; Ng, K. W.; Fan, L.; and Yang, Q. 2021. Protecting intellectual property of generative adversarial networks from ambiguity attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3630–3639.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Köpf, A.; Yang, E. Z.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H. M.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E. B.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*

2019, December 8-14, 2019, Vancouver, BC, Canada, 8024–8035.

Peng, H.; Huang, S.; Geng, T.; Li, A.; Jiang, W.; Liu, H.; Wang, S.; and Ding, C. 2021. Accelerating transformer-based deep learning models on fpgas using column balanced block pruning. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 142–148. IEEE.

Putra, A.; Adiono, T.; Sutisna, N.; Syafalni, I.; and Mulyawan, R. 2021. Hardware dataflow for convolutional neural network accelerator. In *2021 International Symposium on Electronics and Smart Devices (ISESD)*, 1–6. IEEE.

Quan, Y.; Teng, H.; Chen, Y.; and Ji, H. 2020. Watermarking deep neural networks in image processing. *IEEE transactions on neural networks and learning systems*, 32(5): 1852–1865.

Rouhani, B. D.; Chen, H.; and Koushanfar, F. 2018. DeepSigns: A Generic Watermarking Framework for IP Protection of Deep Learning Models. *CoRR*, abs/1804.00750.

Shinn, C. 2019. tiny-tpu. https://github.com/cameronshinn/tiny-tpu. Accessed: 2022-03-17.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Bengio, Y.; and LeCun, Y., eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Sze, V.; Chen, Y.-H.; Emer, J.; Suleiman, A.; and Zhang, Z. 2017. Hardware for machine learning: Challenges and opportunities. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*, 1–8. IEEE.

Talib, M. A.; Majzoub, S.; Nasir, Q.; and Jamal, D. 2021. A systematic literature review on hardware implementation of artificial intelligence algorithms. *The Journal of Supercomputing*, 77(2): 1897–1938.

Thompson, N. C.; Greenewald, K.; Lee, K.; and Manso, G. F. 2021. Deep Learning's Diminishing Returns: The Cost of Improvement is Becoming Unsustainable. *IEEE Spectrum*, 58(10): 50–55.

Tramèr, F.; Zhang, F.; Juels, A.; Reiter, M. K.; and Ristenpart, T. 2016. Stealing Machine Learning Models via Prediction APIs. In Holz, T.; and Savage, S., eds., *Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12*, 601–618. USENIX Association.

Uchida, Y.; Nagai, Y.; Sakazawa, S.; and Satoh, S. 2017. Embedding Watermarks into Deep Neural Networks. In *International Conference on Multimedia Retrieval (ICMR), Bucharest, Romania, June 6-9*, 269–277. ACM.

Wei, J.; Zhang, Y.; Zhou, Z.; Li, Z.; and Al Faruque, M. A. 2020. Leaky DNN: Stealing deep-learning model secret with GPU context-switching side-channel. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, 125–137. IEEE.

Wightman, R. 2019. PyTorch Image Models. https://github.com/rwightman/pytorch-image-models. Accessed: 2022-03-17.

Wu, B.; and Ghanem, B. 2019. ℓ_p-Box ADMM: A Versatile Framework for Integer Programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 41(7): 1695–1708.

Xue, M.; Wang, J.; and Liu, W. 2021. DNN Intellectual Property Protection: Taxonomy, Attacks and Evaluations (Invited Paper). In Chen, Y.; Zhirnov, V. V.; Sasan, A.; and Savidis, I., eds., *Great Lakes Symposium on VLSI, USA, June 22-25*, 455–460. ACM.

Yang, P.; et al. 2021. Robust watermarking for deep neural networks via bi-level optimization. In *Intl. Conference on Computer Vision*, 14841–14850. IEEE/CVF.

Ye, G.; Gao, J.; Xie, W.; Yin, B.; and Wei, X. 2022. Deep Boosting Robustness of DNN-based Image Watermarking via DBMark. *arXiv preprint arXiv:2210.13801*.

Yu, H.; Yang, K.; Zhang, T.; Tsai, Y.; Ho, T.; and Jin, Y. 2020. CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples. In *Network and Distributed System Security Symposium San Diego, California, USA, February 23-26, 2020*. The Internet Society.

Zhang, J.; et al. 2018. Protecting Intellectual Property of Deep Neural Networks with Watermarking. In *Asia Conference on Computer and Communications Security (AsiaCCS), Incheon, Republic of Korea, June 04-08*, 159–172. ACM.

Zhang, W.; Cui, W.; Jiang, F.; Yang, C.; and Li, R. 2021. Protecting the Ownership of Deep Learning Models with An End-to-End Watermarking Framework. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 76–82. IEEE.

Zhao, B.; and Lao, Y. 2022. CLPA: Clean-Label Poisoning Availability Attacks Using Generative Adversarial Nets. In *Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI)*.

Zhu, L.; Li, Y.; Jia, X.; Jiang, Y.; Xia, S.-T.; and Cao, X. 2021. Defending against Model Stealing via Verifying Embedded External Features. In *ICML 2021 Workshop on Adversarial Machine Learning*.

Zou, Z.; Gong, B.; and Wang, L. 2021. Anti-Neuron Watermarking: Protecting Personal Data Against Unauthorized Neural Model Training. *arXiv preprint arXiv:2109.09023*.