Reliable Hardware Watermarks for Deep Learning Systems

Joseph Franklin Clements and Yingjie Lao, *Senior Member, IEEE*

***Abstract*— Recent successes in deep learning have indicated that hardware technologies will play a prominent role in future deep learning industries and applications. In light of their value, researchers have recognized that deep neural networks (DNNs) and other deep learning intellectual properties (IPs) can be easily pirated, especially in undefended settings. While multiple avenues of defending deep learning systems have been identified, watermarks are particularly valuable as they allow IP theft to be identified and remedied when it occurs. However, such defenses have yet to be considered for defending the hardware platforms running the deep learning systems. This article presents the first framework for applying watermarks toward defending deep-learning hardware accelerators from piracy, called DeepHardMark. The proposed methodology embeds modifications into the functional blocks of deep-learning hardware accelerators to act as a watermark signature. These modifications produce targeted alterations to the execution of key DNNs on corresponding key samples, which identifies the hardware. We optimize this methodology to simultaneously minimize the impact of the watermark embedding on both the hardware and algorithmic components of the deep learning system making the watermark unobtrusive and challenging to detect. Our experimental evaluations demonstrate the feasibility of embedding the proposed modifications into typical hardware designs and in various deep-learning scenarios.**

***Index Terms*— Deep learning, hardware accelerators, intellectual property (IP), security, watermarks.**

NOMENCLATURE

DNN Models	
$F(\cdot)$	Generic DNN.
$F_k(\cdot)$	Key DNN.
$F^{\delta}(\cdot)$	DNN executed under a perturbation (δ).
Dataset	
$\{\mathbf{x}_i, \mathbf{y}_i\}$	/Input sample, ground truth output pairs.
$\{\mathbf{x}_k, \mathbf{y}_k\}_1^K$	K pairs of key sample, target output pairs.
Model Perturbationss	
$\widehat{\delta_k}$	Block constrained perturbation.
δ_k	Operation reduced perturbation.
μ_k	Hardware modification set.
Binary Masks	
$\mathbf{H}$	Hardware mask.
$\mathbf{B}$	Block selection mask.
$\mathbf{R}$	Reduced selection mask.
Algorithmic Parameters	
ψ	Cardinality constraint.
ϵ_δ, ϵ_ψ, ϵ_B	Step sizes.
T_δ, T_ψ, T_B	Max iteration count.

Manuscript received 18 August 2023; revised 12 December 2023; accepted 9 January 2024. Date of publication 9 February 2024; date of current version 22 March 2024. This work was supported by the National Science Foundation under Award 2047384 and Award 2247620. *(Corresponding author: Yingjie Lao.)*

Joseph Franklin Clements is with the Integrated Products Division, Applied Research Associates, Albuquerque, NM 87110 USA (e-mail: jclements@ara.com).

Yingjie Lao is with the Department of Electrical and Computer Engineering, Tufts University, Medford, MA 02155 USA (e-mail: yingjie.lao@tufts.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2024.3360240.

Digital Object Identifier 10.1109/TVLSI.2024.3360240

I. INTRODUCTION

AS DEEP learning continues to develop, and there are increasing incentives to deploy neural networks to dedicated hardware platforms for improved performance and efficiency [1]. Well-optimized hardware can give deep learning providers a competitive edge and open the door to new markets, like edge computing [2]. FPGA and ASIC solutions can often provide performance, and efficiency boosts beyond traditional general-purpose hardware [3]. Despite the significant production costs, these specialized hardware platforms are valuable intellectual property (IP) that can produce a significant return on investment [4]. Unfortunately, the modern globalized supply chain faces numerous security challenges, including piracy, misuse, overproduction, reverse engineering, and malicious modification [5]. Protecting deep-learning hardware designs from IP theft is a critical concern [6].

While techniques like logic locking [7] and design-for-trust [8] can mitigate IP theft, these techniques are not infallible, and innovative adversarial approaches are often introduced to bypass such defenses [9]. As such, detection methods, such as watermarks, become the final measure in defending an IP from theft [10]. Watermarking is the practice of embedding a signature into an IP such that it can identify the rightful owner upon fraudulent usage. The recent progress of deep learning has driven developers to extend this concept to protecting deep neural networks (DNNs) and other valuable algorithmic IPs [11], [12]. A typical deep learning watermarking scheme embeds an intentional and nonadversarial backdoor into a protected model. The backdoor introduces functionality into the model that deviates from its intended task and can be identified as unique to the model, i.e., the watermark signature. However, these methodologies have been conventionally applied to protect only the software IPs. Inspired by recent works into deep-learning hardware backdoors and Trojan-inspired hardware watermarks [13], our prior conference paper DeepHardMark [14] became the

first work to apply such techniques to protect deep-learning hardware IPs from piracy.

This work presents an extension of the DeepHardMark [14] framework: DeepHardMark$^+$. This framework improves the original work by introducing a gradient descent-based methodology for the intrablock perturbation reduction phase. This improvement allows us to abandon the resource-heavy search-based algorithm previously used in DeepHardMark to extend the algorithm to larger, more complex architectures, including state-of-the-art transformer-based models. Furthermore, we incorporate algorithmic optimizations into DeepHardMark, allowing the automatic initialization of key parameters to improve the original algorithm's performance. We demonstrate the significance of these improvements in the experimental evaluation by performing watermark embedding on hardware intended for a broad range of deep-learning scenarios. In addition to these contributions, we solidify key components of the original DeepHardMark algorithm, including a more rigorous technical description linking the algorithmic perturbations found by the DeepHarMark algorithms and the hardware modifications it embeds. Our contributions are summarized below.

1) This article presents, DeepHardMark$^+$, which improves upon the first hardware Trojan-inspired watermarking framework by implementing a novel gradient descent-based algorithm for finding the operation reduction perturbation.
2) In addition, we utilize a methodology for minimizing the target cardinality constraint during optimization to ensure the algorithm reliably converges to the optimal solution.
3) We present the first mathematical definition of the hardware modifications which provides a general blueprint for the developer to produce the hardware modifications.
4) We experimental demonstrate that our methodology minimizes the embedded watermark's impact from both the hardware and algorithmic perspectives while successfully embedding the hardware watermark on a broad range of modern architectures.

We organize this article as follows. Section II provides background on current trends in deep-learning hardware development, watermark embedding frameworks, and backdoor injection. We then present the threat model for the piracy violation of deep-learning hardware and propose the DeepHardMark$^+$ watermark framework as a defense against such attacks in Section III. We follow this, in Section V, with experimental evaluation demonstrating the efficacy of DeepHardMark$^+$. Our investigations cover the impact of both DeepHardMark and DeepHardMark$^+$ on a broad range of deep learning scenarios and open-source hardware accelerators. In Section VI, we discuss the broader impact and future directions. Finally, we end the work in Section VII with a brief conclusion.

II. Background

A. Developing Novel Hardware for Deep Learning

Deep learning systems are often determined to be superior to traditional approaches for many application domains. However, the complexity of these models usually requires increased resource utilization, including data storage, power draw, and computation time [15]. As such, hardware accelerators for DNNs have seen a resurgence in recent years [16]. While general processors, like GPUs, have enabled the widespread availability of deep learning, there is an increasing demand for low-latency or low-power technologies, specifically in edge computing and IoT, where such resources are limited [17]. Premium hardware accelerators maximize the utilization of hardware resources when executing a DNN model through efficient memory hierarchies and dataflows, which determines when and where operations are computed and where data is stored or reused [18]. By carefully considering the specific target deep learning system and its intended usage, systems designers can generate highly effective devices that excel well beyond the performance possible with general purpose devices [19].

B. Injecting Backdoors in Hardware Designs

Hardware Trojans are modifications injected into the intended circuit designs by an adversary either during design or fabrication. These modifications introduce malicious behavior into the design, such as stealing secured information or manipulating software executed on the device [20]. The Trojan behaviors are often designed to only be activated when unique input conditions are satisfied, making them very difficult to defend against even with modern techniques, especially without the Trojan-free design [21]. Recent developments have revealed that deep learning systems are also susceptible to manipulation through their hardware platform [22]. Hardware Trojans are a practical approach for injecting backdoors into deep learning models [23]. In much the same way that backdoors can be repurposed as a watermark to protect deep learning algorithmic IPs, a recent study has demonstrated a designer can similarly leverage the hardware Trojans to embed watermarks into hardware IPs [13]. DeepHardMark framework integrates this perspective from the hardware domain with an understanding of deep-learning algorithmic IP to effectively embed watermarks into deep-learning hardware designs.

C. Defending Deep Learning IPs From Piracy

Watermarking is a technique deployed as a countermeasure to IP theft. Conventional approaches in multimedia often introduce near-invisible distortions to an image that identifies its owner [24]. This hidden signature allowed the rightful owner to claim fraudulent usage of the IP. Recently concern over the ease with which adversaries can steal deep learning models has motivated researchers to extend the concept of watermarking to DNNs [25]. A common approach for watermarking neural network models is leverage model poisoning [26] and backdoor [27] attacks to embed abnormal and hidden functionality into the neural network [28]. The rightful owner can reveal this hidden functionality as a proof of the model's origin [29]. Recently, similar backdoor-based watermarking techniques have been extended to protect deep learning datasets [30].

A similar identifying methodology, fingerprinting, has also been investigated for defending DNN models recently [31]. These techniques attempt to identify DNN models' characteristics that developers can use to verify IP ownership without altering the model's functionality [32]. However, these previous works have exclusively been deployed for defending the algorithmic IPs in deep learning, such as the DNN model. In this environment, hardware has primarily been used to protect algorithmic IPs running on a defended device [33]. Due to the high value of deep-learning hardware and the potential for adversaries to access hardware designs through horizontal supply chains, a recent work developed the DeepHardMark framework, which extends the protection of watermarking techniques to deep-learning hardware IPs [14].

D. Watermark-Free Defenses Against Hardware Piracy

Many methods of defending Hardware designs from piracy exist. Obfuscation-based protections attempt to hide the functionality of a hardware design. For example, designers can utilize techniques for encrypting or modifying RTL sources, so it is incomprehensible to protect higher level designs [34], [35]. Logic locking designs hardware in such a way that the device only works when specific keys are applied to the device [36], [37]. Design for trust techniques can be used to set up supply chains such that single point of entry attacks do not compromise the security of an IP [38]. However, no method of defending hardware IPs from piracy is perfect, and designers benefit from providing multiple layers of security for their designs. Watermarks differ from these approaches in that they serve as an authentication-based method for identifying devices after theft.

The process of watermarking finite state machines (FSMs) has been extensively explored. These methods add new states [39], abnormal I/O sequences [40], or state encodings [41] to embed information into the FSM. This effectively embeds information into the unused portions of an FSM that can later be extracted and used to identify the system. However, deep learning accelerators are composed of large arrays of computational blocks computing well-understood mathematical operations. Such operations do not contain such unused cases but are typically expected to accurately compute their operation for all input combinations. Deep learning is a special case, as it is well known to be highly robust to most noise. This allows for the unique opportunity of embedding modifications into the core functional blocks of the deep learning system, providing a very stealthy and effective watermark.

III. Problem Definition

A. Threat Model

Globalization in semiconductor supply chains often outsources manufacturing applications to third parties. This procedure provides an attack vector threat actors can utilize to access critical manufacturing applications. This work considers a setting where an adversary has used this access to pirate a valuable deep-learning hardware accelerator and distribute the device for use in their own applications. The goal of the proposed watermark defense is to embed a watermark signature into the hardware such that the rightful owner can prove their right to the device. The owner must embed the signature before manufacturing without any knowledge of the adversary. As stated in prior works [10], we can assume that the adversary does not have access to a behavioral description of the design.

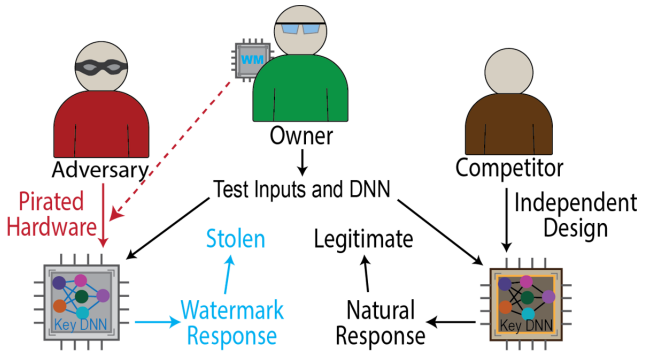

Fig. 1. DeepHardMark provides the first framework for watermarking deep-learning hardware. It functions by embedding modifications in the design, which the owner can activate with a key DNN/key sample pair producing an abnormal behavior. The distinction between this watermark response and the natural response of watermark-free designs can be used to prove ownership over pirated hardware.

Furthermore, the signature must be identifiable without direct access to the device's internal functioning and be reliably accessed through remote API calls. We assume, however, that these API calls allow the user freedom in selecting the model and inputs computed by the model. This requirement is practical in most deep learning applications, which observe the best performance when finely tuned for specific downstream tasks. This threat model is consistent with the literature of hardware watermarking [10]. As seen in Fig. 1, DeepHardMark extends backdoor-inspired deep-learning watermarks to the hardware domain, establishing the first watermark-based defense for deep-learning hardware accelerators against piracy.

B. Proposed Hardware Watermarking

The DeepHardMark algorithm connects a hardware accelerator to deep learning algorithmic components held by the rightful owner. The algorithm begins by selecting the algorithmic components: a key DNN, $F_k(\cdot)$, and K key samples, $\{\mathbf{x}_k\}_1^K$ used in generating the watermark. A target model functionality, $\mathbf{y}_k$, is selected for the key samples that can be identified as abnormal behavior for the model, i.e., $F_k(\mathbf{x}_k) \neq \mathbf{y}_k$. The watermarking objective is to embed a signature into the system such that it produces abnormal behavior under the key samples. Ideally, the watermark should only alter the system for the key samples

$$F_k^{\delta}(\mathbf{x}) = \begin{cases} \mathbf{y}_k, & \text{when } \mathbf{x} == \mathbf{x}_k \\ F_k(\mathbf{x}), & \text{otherwise.} \end{cases} \tag{1}$$

We utilize $F_k^{\delta}(\mathbf{x})$ here to identify the signature embedded system, where δ represents the specific alteration to the system. Prior works in the algorithmic perspective have performed watermark signature embeddings by altering the algorithmic components directly.

The signature must be embedded within the hardware design to provide proper protection for the hardware. Deep-learning hardware designs are composed of arrays of functional blocks, each targeted at executing specific operations with high throughput. The owner selects a target array of functional blocks and determines the operations in the key DNN, which will be computed on these blocks according to the device's hardware mapping scheme. We can use this mapping to produce a hardware mask, $\mathbf{H} \in \{0,1\}^{O\times N}$, that identifies which of the N functional blocks in the hardware each of the O operations will be executed on. Modifying these functional blocks can slightly alter the execution of the operations as they are executed and change the outcome of inference. Prior works have shown that such modifications can use this to produce targeted changes in a model's behavior [23]. However, the magnitude of modifications to these functional blocks must be minimal to ensure a negligible impact on the hardware overhead and change in the hardware's functionality. As such, we introduce a binary mask, $\mathbf{B} \in \{0,1\}^{O}$, to minimize the number of functional blocks modified. We define a deep learning system modified through the hardware as

$$F_k^{\delta_k}(\mathbf{x}) = \begin{cases} \mathbf{y}_k, & \text{when } \mathbf{x} == \mathbf{x}_k \\ F_k(\mathbf{x}), & \text{otherwise} \end{cases} \tag{2}$$

where δ_k describes an algorithmic perturbation on the model when executed by the target functional blocks, for producing the watermark signature.

From the deep learning perspective, we can describe δ_k as a perturbation of the model's latent representations, $\mathbf{h}_l = \mathbf{h}_l + \delta_{k,l}$ for all layers, l. From the hardware perspective, δ_k is generated by some set of modifications, μ_k, embedded in the design. Our approach to producing these modifications is to utilize two small combinational circuits for each perturbed operation. The first observes the inputs to a functional block. When it detects a rare internal state indicating the target operation is being executed, it sends a signal to the second circuit activating it. Once activated, the second circuit flips bits on the output that would produce the desired perturbation to the operation. δ_k is an algorithmic approximation of such a change to the deep learning system when processing the key sample. However, this is beneficial as it implies methods for detecting and bypassing watermarks from the algorithmic perspective do not directly transfer to those in the hardware.

Embedding the modification set, μ_k, into the hardware establishes an observable difference between the key DNN's behavior on watermark-free hardware and modified hardware. The device's rightful owner can reveal the presence of this signature by first demonstrating the similarity of the hardware to watermark-free counterparts. Once the consistency of operation is established, the owner can then reveal their key DNN and key samples and demonstrate the abnormal behavior they produce on the modified hardware. As the behavioral change is tied to the hardware and does not modify any algorithmic components, this establishes a link between the hardware design and the owner's key DNN and key sample. This verification procedure follows a scheme similar to those used to protect various algorithmic components, including models [42] and datasets [30]. DeepHardMark is the first methodology that extends this understanding to deep-learning hardware accelerators.

IV. Methodology

A. Block Constrained Perturbations

We begin by defining the optimization problem seen in (3) to produce $\widehat{\delta_k}$, the block-constrained perturbation. This problem utilizes the relationship $\widehat{\delta_k} = \delta \odot \mathbf{BH}$ to simultaneously embed the watermark signature and minimize the functional blocks targeted for modification

$$\begin{aligned} &\underset{\delta,\mathbf{B}}{\text{minimize}}\ L(F_k^{\delta\odot\mathbf{BH}}(\mathbf{x}_k), \mathbf{y}_k) \\ &\text{s. t. } \mathbf{1}^T\mathbf{B} < \psi,\ \ \mathbf{B} \in \{0,1\}. \end{aligned} \tag{3}$$

We use L to represent a loss function, such as cross-entropy loss, that quantifies the watermarking objective for a target output, $\mathbf{y}_k$. $\mathbf{1}^T\mathbf{B} < \psi$ is a cardinality constraint that defines an upper bound on the magnitude of $\mathbf{B}$.

The selection of ψ is critical to the strength of the watermark embedding as it contributes to determining the number of functional blocks targeted by the algorithm. DeepHardMark [14] applies a brute force method of determining ψ by decreasing constant selections for ψ until the algorithm cannot find a solution. However, such strategies result in inferior solutions. In the DeepHardMark$^+$ algorithm, we improve the selection of this parameter by integrating it into the algorithm allowing it to settle into a more desirable solution.

We do this by initializing ψ to the largest constraint possible, $\psi = |\mathbf{B}|$, i.e., the total number of functional blocks in the hardware. Then, after updating $\mathbf{B}$ and δ, we verify that a valid solution was found by asserting that $F_k^{\delta\odot\mathbf{BH}}(\mathbf{x}_k) == \mathbf{y}_k$. If the algorithm finds a valid solution, we record it as $\psi^M = \psi$. Then, tighten the constraint by setting $\psi = \lceil \psi * \epsilon_\psi \rceil$ where $\epsilon_\psi \in (0,1)$ determines the rate of decrease. If a solution is not found, we instead relax the constraint by perturbing ψ toward ψ^M, the lowest observed value which produced a valid solution, using $\psi = \lceil \psi + (1/T_\psi) * (\psi^M - \psi) \rceil$. The cooling temperature, T_ψ, is a variable used to control how swiftly ψ returns to ψ^M. This framework allows us to dynamically decrease the cardinality constraint as the algorithm converges and then relax the constraint when it becomes too difficult for a solution to be found.

We employ the recently developed ℓ_p-box alternating direction method of multipliers (ℓ_p-ADMMs) [43] to solve (3). We defer the details of the derivation of this algorithm to DeepHardMark [14] and present just the optimization procedure here. Solving (3) is conducted by alternating between updating δ and $\mathbf{B}$. We begin by first fixing $\mathbf{B}$ to $\mathbf{1}$ and solving for δ using the following equation:

$$\delta = \delta - \epsilon_\delta \left[\frac{\partial L(F_k^{\delta\odot\mathbf{BH}}(\mathbf{x}_k), \mathbf{y}_k)}{\partial \delta} \right]. \tag{4}$$

Here, ϵ_δ is a step size, a hyper-parameter used to control convergence speed. We perform this update step iteratively until the signature is embedded, verified by $F_k^{\delta\odot\mathbf{BH}}(\mathbf{x}_k) == \mathbf{y}_k$.

Algorithm 1 Block Constrained Perturbations

Require: $L(\cdot)$, $\mathbf{H}$, $F_k(\cdot)$, $\mathbf{x}_k$, $\mathbf{y}_k$
Hyperparameters: ϵ_δ, ϵ_B, ϵ_ψ, T_δ, T_ψ, T_B
Ensure: $F_k(\mathbf{x}_k) \neq \mathbf{y}_k$
$\mathbf{B} = \mathbf{1}$; $\delta = \mathbf{0}$; $\psi^M = \psi = |\mathbf{B}|$
while $F_k^{\delta \odot \mathbf{BH}}(\mathbf{x}_k) \neq \mathbf{y}_k$ **do**
  for $i \in [1, T_\delta]$ **do**
    $\delta = \delta - \epsilon_\delta \left[\frac{\partial L(F_k^{\delta \odot \mathbf{BH}}(\mathbf{x}_k), \mathbf{y}_k)}{\partial \delta}\right]$
  end for
  if $F_k^{\delta \odot \mathbf{BH}}(\mathbf{x}_k) == \mathbf{y}_k$ **then**
    $\psi = \lceil \psi * \epsilon_\psi \rceil$; $\psi^M = \psi$;
  else
    $\psi = \lceil \psi + \frac{1}{T_\psi} * (\psi^M - \psi) \rceil$
  end if
  $\mathbf{Z}_1 = \mathbf{Z}_2 = \mathbf{Z}_3 = \mathbf{1}$; $\psi_1 = |\mathbf{B}|$
  for $i \in [1, T_B]$ **do**
    $\mathbf{B} = \mathbf{B} - \epsilon_B \left[\frac{\partial \mathcal{L}}{\partial \mathbf{B}}\right]$ # $\mathcal{L}$ is defined in Equation (8)
    $\psi_i = \psi_{i-1} - \frac{1}{T_B} * (|\mathbf{B}| - \psi)$
    Update the dual parameters using Equation (9)
  end for
end while
$\widehat{\delta}_k = \delta \odot \mathbf{BH}$
return $\widehat{\delta}_k$

Then, for a fixed value of δ, we solve for $\mathbf{B}$ using the update step

$$\mathbf{B} = \mathbf{B} - \epsilon_B \left[\frac{\partial \mathcal{L}}{\partial \mathbf{B}}\right] \tag{5}$$

where

$$\frac{\partial \mathcal{L}}{\partial \mathbf{B}} = \frac{\partial L(F_k^{\delta \odot \mathbf{BH}}(\mathbf{x}_k), \mathbf{y}_k)}{\partial \mathbf{B}} + \zeta_1(\mathbf{B} - \mathbf{S}_1) + \mathbf{Z}_1 + \zeta_2(\mathbf{B} - \mathbf{S}_2) + \mathbf{Z}_2 + [\zeta_3(\mathbf{1}^T\mathbf{B} - \psi) + \mathbf{Z}_3]\mathbf{1}. \tag{6}$$

Here, $\mathbf{S}_1 = \max(\min(\mathbf{B}, \mathbf{1}), \mathbf{0})$ and $\mathbf{S}_2 = ((M)^{1/2}/2)(\mathbf{B} - 0.5(\mathbf{1})/\|\mathbf{B} - 0.5(\mathbf{1})\|) + (1/2)(\mathbf{1})$. $\mathbf{Z}_1 \in \mathbb{R}^M$, $\mathbf{Z}_2 \in \mathbb{R}^M$, and $\mathbf{Z}_3 \in \mathbb{R}^1$ are dual variables initialized to $\mathbf{1}$ with corresponding penalty parameters: ζ_1, ζ_2, and ζ_3, respectively. The penalty parameters balance the terms of (6).

The cardinality constraint, ψ, is used in (3) as a target for the number of functional blocks to be selected by $\mathbf{B}$. However, while solving for $\mathbf{B}$, we can further improve our control over the convergence rate by targeting a decreasing schedule of ψ_i converging to ψ. To accomplish this, we define the rule below for migrating toward ψ while optimizing $\mathbf{B}$

$$\psi_0 = |\mathbf{B}|$$
$$\psi_i = \psi_{i-1} - \frac{1}{T_B} * (|\mathbf{B}| - \psi) \tag{7}$$

where T_B refers to the number of iterations used in solving for $\mathbf{B}$. We then rewrite (6) with ψ_i

$$\frac{\delta \mathcal{L}}{\delta \mathbf{B}} = \frac{\delta L(F_k^{\delta \odot \mathbf{BH}}(\mathbf{x}_k), \mathbf{y}_k)}{\delta \mathbf{B}} + \zeta_1(\mathbf{B} - \mathbf{S}_1) + \mathbf{Z}_1 + \zeta_2(\mathbf{B} - \mathbf{S}_2) + \mathbf{Z}_2 + [\zeta_3(\mathbf{1}^T\mathbf{B} - \psi_i) + \mathbf{Z}_3]\mathbf{1}. \tag{8}$$

This update produces a more elastic downward pressure on $\mathbf{B}$, which allows the algorithm to slowly settle into a valid solution over multiple iterations of the algorithm, often contributing to a better solution.

Finally, we update the dual variables in each iteration with the following:

$$\mathbf{Z}_1 = \mathbf{Z}_1 + \zeta_1(\mathbf{B} - \mathbf{S}_1)$$
$$\mathbf{Z}_2 = \mathbf{Z}_2 + \zeta_2(\mathbf{B} - \mathbf{S}_2)$$
$$\mathbf{Z}_3 = \mathbf{Z}_3 + \zeta_3(\mathbf{1}^T\mathbf{B} - \psi_i). \tag{9}$$

Using this procedure, summarized in Algorithm 1, we are able to determine $\widehat{\delta}_k = \delta \odot \mathbf{BH}$. This block-constrained perturbation can embed the watermark signature into the key DNN while altering only the operations executed on a minimal set of functional blocks in the target hardware.

B. Intrablock Perturbation Reduction

The block-constrained perturbation, $\widehat{\delta}_k$, is targeted at minimizing the number of hardware blocks perturbed by the watermarking algorithm. However, $\widehat{\delta}_k$ is not optimized to constrain the total perturbations within each block. Thus, it is likely that redundant perturbations that contribute little to the watermark's performance are contained in $\widehat{\delta}_k$. In the next step in the algorithm, we remove these redundant perturbations by finding a minimal subset of $\widehat{\delta}_k$, which still produces the desired watermark signature.

1) Search-Based Approach: We can mathematically define $\delta_k = \mathbf{R} \odot \widehat{\delta}_k$, an operation reduced perturbation, where $\mathbf{R} \in \{0, 1\}^N$ is the reduced selection mask which specifies which perturbations to retain. We accomplished this using the optimization problem

$$\underset{\mathbf{R}}{\text{minimize}} \ \|\mathbf{1}^T\mathbf{R}\| \quad \text{s. t. } F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k) == \mathbf{y}_k \tag{10}$$

which is solved by iteratively selecting the elements of $\widehat{\delta}_k$ with the greatest impact on the objective function and activating them with $\mathbf{R}$. DeepHardMark utilizes the brute force search algorithm presented in Algorithm 2 to accomplish this.

The search algorithm begins with two sets: $\mathbf{R}_\delta = \mathbf{0}$ and $\mathbf{R}_N = \{\mathbf{R}_n | \ \|\mathbf{R}_n\|_\infty == 1, \ \mathbf{1}^T\mathbf{R}_n == 1, \ \mathbf{R}_n \odot \widehat{\delta}_k \neq 0\}$. We can understand $\mathbf{R}_N$ as the set of all significant single-bit variations of $\mathbf{R}$. The algorithm's goal is to iteratively incorporate members from $\mathbf{R}_N$ into $\mathbf{R}_\delta$ by selecting the most effective choice at each step of the algorithm. We do this by generating the Cartesian sum of both sets and determining which the choice of $\mathbf{R}_d \in \mathbf{R}_\delta$, and $\mathbf{R}_n \in \mathbf{R}_N$ best minimizes the loss function, $L(F_k^{(\mathbf{R}_d + \mathbf{R}_n) \odot \widehat{\delta}_k}(\mathbf{x}_k), \mathbf{y}_k)$. These choices are then used to populate $\mathbf{R}_\delta$ during the next iteration of the algorithm, iteratively increasing the number of bits selected by the members of $\mathbf{R}_\delta$. Furthermore, we incorporate the beam search techniques by keeping the C best choices for $\mathbf{R}_\delta$ rather than only the best. Once an $\mathbf{R}$ which produces the desired watermark signature is found, we can compose the operation reduced perturbation as $\delta_k = \mathbf{R} \odot \widehat{\delta}_k$.

Algorithm 2 Search-Based Intra-Block Reduction

Require: $\widehat{\delta}_k$, $L(\cdot)$, $F(\cdot)$, C
$\mathbf{R}_\delta = \{\mathbf{0}\}$
$\mathbf{R}_N = \{\mathbf{R}_n | \ \|\mathbf{R}_n\|_\infty == 1,\ \mathbf{1}^T\mathbf{R}_n == 1,\ \mathbf{R}_n \odot \widehat{\delta}_k \neq 0\}$
while $F_k^{\mathbf{R}_d \odot \widehat{\delta}_k}(\mathbf{x}_k) \neq \mathbf{y}_k \ \forall\ \mathbf{R}_d \in \mathbf{R}_\delta$ **do**
  $\mathbf{R}_\rho = \{\mathbf{R}_d + \mathbf{R}_n | \mathbf{R}_d \odot \mathbf{R}_n == \mathbf{0}, \mathbf{R}_d \in \mathbf{R}_\delta, \mathbf{R}_n \in \mathbf{R}_N\}$
  Loss $= [\ ]$
  for $\mathbf{R}_r \in \mathbf{R}_\rho$ **do**
    $l_r = L(F_k^{\mathbf{R}_r \odot \widehat{\delta}_k}(\mathbf{x}_k), \mathbf{y}_k)$
    Loss.append($(\mathbf{R}_r,\ l_r)$)
  end for
  sort_by_loss(**Loss**)
  $\mathbf{R}_\delta = \{$**Loss**$[0 : C-1][0]\}$
end while
$\mathbf{R} = \underset{\mathbf{R}_d \in \mathbf{R}_\delta}{\text{argmin}}\ L(F_k^{\mathbf{R}_d \odot \widehat{\delta}_k}(\mathbf{x}_k), y_k)$
return $\mathbf{R}$

2) Gradient Decent-Based Approach: The search-based approach to finding $\delta_k = \mathbf{R} \odot \widehat{\delta}_k$ is a computationally expensive process compounded by the need to maintain a list of the C-best solutions to decrease the likelihood that the algorithm converges suboptimally. Using this algorithm limits DeepHardMark's usefulness when defending larger scale models with many parameters. For the DeepHardMark$^+$ algorithm, we utilize an alternative gradient-based methodology for finding δ_k.

We begin by recognizing that we know a solution to $F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k) == \mathbf{y}_k$ exists when $\mathbf{R} \odot \widehat{\delta}_k = \widehat{\delta}_k$, i.e., $\mathbf{R} = \mathbf{1}$. However, this stage of the algorithm aims to minimize $\|\mathbf{1}^T\mathbf{R}\|$, the number of operations targeted by the hardware modifications. Using a loss function, L, we can redefine the objective from (10) in the form

$$\begin{aligned} &\underset{\mathbf{R}}{\text{minimize}}\ L(F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k), \mathbf{y}_k) \\ &\text{s. t. } \|\mathbf{1}^T\mathbf{R}\| < \varepsilon \end{aligned} \tag{11}$$

where ε is an upper bound on the size of $\mathbf{R}$. Finding the optimal $\mathbf{R}$ implies finding the tightest bound on ε with a valid solution to (11). Solving for (11) is analogous to finding the sparse solution to the objective function: $L(F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k), \mathbf{y}_k)$.

To solve this, we perform an iterative algorithm using the gradient information. The proposed methodology can be seen in Algorithm 3. We begin with the initialization $\mathbf{R} = \mathbf{0}$. We then iteratively activate bits in $\mathbf{R}$ by first calculating $L(F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k), \mathbf{y}_k)$ using $\mathbf{R}$. $\mathbf{R}$ is used as a mask to turn off individual elements of the unreduced perturbation, $\widehat{\delta}_k$. The gradient of L with respect to $\mathbf{R}$ is a valid local approximation of the impact switching bits in the mask on the loss. As such, we find the element with maximal effect on the loss with $M = \text{argmax}((\delta L/\delta \mathbf{R}) \odot \mathbf{BH})$. We can then iteratively activate those bits in $\mathbf{R}$ until we arrive at a valid solution for (11). Once a $\mathbf{R}$ is selected, which produces the desired mapping $F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k) == \mathbf{y}_k$ this mask can be used to extract a subset of $\widehat{\delta}_k$ which produces the watermark signature.

However, as highlighted by the sparse adversarial example literature [44], such methods often introduce unnecessary or redundant elements to δ_k. To ensure the selection of the best mask, we perform additional rounds of optimization by activating features with minimal impact on the loss function, identified by $m = \text{argmin}((\delta l/\delta \mathbf{R}) \odot \mathbf{BH})$. Then, we deactivate elements until the watermark signature is removed, i.e., $F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k) \neq \mathbf{y}_k$. We repeatedly activate the necessary and deactivate unnecessary elements of $\mathbf{R}$ until both branches of the algorithm toggle the same set of features. This process ensures that we find a $\mathbf{R}$ which satisfies (11) for a minimal choice of ε.

Algorithm 3 Gradient-Based Intra-Block Reduction

Require: $\widehat{\delta}_k$, $L(\cdot)$, $F(\cdot)$, $\mathbf{x}_k$, $\mathbf{y}_k$
$\mathbf{R} = \mathbf{0}$
while $M \neq m$ **do**
  while $F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k) \neq \mathbf{y}_k$ **do**
    $l = L(F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k), \mathbf{y}_k)$
    $M = \text{argmax}(\frac{\delta l}{\delta \mathbf{R}} \odot \mathbf{BH})$
    $\mathbf{R}[M] = 1$
  end while
  while $F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k) == \mathbf{y}_k$ **do**
    $l = L(F_k^{\mathbf{R} \odot \widehat{\delta}_k}(\mathbf{x}_k), \mathbf{y}_k)$
    $m = \text{argmin}(\frac{\delta l}{\delta \mathbf{R}} \odot \mathbf{BH})$
    $\mathbf{R}[m] = 0$
  end while
end while
$\mathbf{R}[M] = 1$
return $\mathbf{R}$

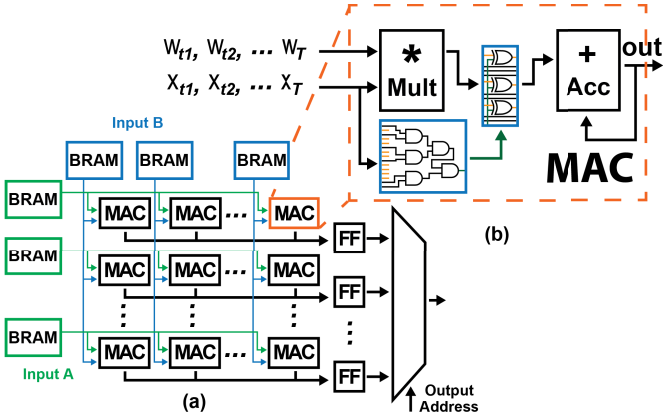

Fig. 2. (a) Convolutional neural network (CNN) hardware accelerator derived from [45]. (b) We can embed small combinational circuits into the hardware blocks of the IP. These circuits detect the target input combinations and flip the corresponding output bits as specified by μ_k.

C. Generating Hardware Modifications

The final stage of the algorithm converts δ_k into a hardware modification set, μ_k, which defines δ_k in terms of modifications that can be embedded in the hardware. As a case study in this article, our implementation embeds small combinational circuits into the target hardware, as shown in Fig. 2. Let op $\in$ Ops be a target operation indicated by δ_k with $O_k = |\delta_k|$. In our example, $\mu_k = \{\mathbf{P}_k, \mathbf{F}_k\}_1^{O_k}$ contains latent representations inputs to an operation, $\mathbf{P}_k$, and bit flip patterns, $\mathbf{F}_k$, that can produce δ_k at the output of an operation when $\mathbf{x}_k$ is being computed.

TABLE I
EFFECTIVENESS AND IMPACT OF EMBEDDING DEEPHARDMARK WATERMARKS

Dataset	Model	Acc%	$ESR\%$	$\lvert\delta_k\rvert\% \pm SD$	ΔAcc% $\pm SD$	ΔFid% $\pm SD$	ΔArea% $\pm SD$
Cifar10	ResNet18	93.02	100.0	0.18 ± 0.09	0.68 ± 0.14	0.12 ± 0.80	0.22 ± 0.39
Cifar100	ResNet18	88.54	100.0	1.29 ± 0.86	0.30 ± 0.42	0.25 ± 0.39	1.72 ± 0.72
ImageNet	ResNet18	89.08	100.0	0.15 ± 0.07	0.67 ± 0.47	0.68 ± 0.47	0.99 ± 0.44

We can determine $\mathbf{P}_k$ and $\mathbf{F}_k$ by analyzing the latent space of $F_k(\cdot)$ under $\mathbf{x}_k$ and δ_k, the operation reduced perturbation. Let $\widehat{\mathbf{P}}_k$ be the latent space representation immediately preceding the layers targeted by the algorithm and $\widetilde{\mathbf{P}}_k = \text{op}(\widehat{\mathbf{P}}_k)$, those immediately following them, under the key sample, $\mathbf{x}_k$. Then, we can define $\mathbf{P}_k$ by masking the inputs to the target op $\in$ Ops using the Reduced Selection Mask, $\mathbf{R}$, from Section IV-B

$$\mathbf{P}_k = \mathbf{R} \odot \widehat{\mathbf{P}}_k. \tag{12}$$

Furthermore, we understanding that we want $\mathbf{F}_k$ to be a mask such that $\mathbf{F}_k \oplus [\mathbf{R} \odot \widetilde{\mathbf{P}}_k]_b = [\mathbf{R} \odot \widetilde{\mathbf{P}}_k + \delta_k]_b$. Thus, we can define $\mathbf{F}_k$ in terms of $\widetilde{\mathbf{P}}_k$ and δ_k as

$$\mathbf{F}_k = [\mathbf{R} \odot \widetilde{\mathbf{P}}_k + \delta_k]_b \oplus [\mathbf{R} \odot \widetilde{\mathbf{P}}_k]_b. \tag{13}$$

Here $\oplus$ is the exclusive-or function. We use the notation $[\cdot]_b$ to denote the use of a binary representation of a value.

Mathematically, we can use $\mathbf{P}_k$ to define a trigger function for the operation, such that

$$\tau_{\text{op}} = \text{comp}([\mathbf{P}_{t,\text{op}}]_b, [\widehat{\mathbf{P}}_{t,\text{op}}]_b) \tag{14}$$

where $[\widehat{\mathbf{P}}_{t,\text{op}}]_b$ is the binary representation of the input of op when a test input, $\mathbf{x}_t$, is computed by the model. Here, $\text{comp}(\cdot,\cdot)$ is an operation that returns 1 if the binary representations of its inputs are the same and 0 otherwise. We can easily translate this function into a simple combinational circuit that activates a trigger signal, τ when it detects a latent representation that matches that of $\mathbf{P}_k$.

The trigger signal can then be fed as input into a second circuit which produces the desired perturbation by flipping bits on the output. We define a mathematical representation of this perturbation functionality for op, using the flip patterns, $\mathbf{F}_k$

$$\text{pert}(\mathbf{F}_k, \widehat{\mathbf{P}}_{t,\text{op}}), \tau_{\text{op}}) = \begin{cases} [\text{op}(\widehat{\mathbf{P}}_{t,\text{op}})]_b \oplus \mathbf{F}_k, & \text{when } \tau_{\text{op}} == 1, \\ [\text{op}(\widehat{\mathbf{P}}_{t,\text{op}})]_b, & \text{otherwise.} \end{cases} \tag{15}$$

This functionality is easily migrated into a minimal combinational circuit which, when embedded into a functional block, works in conjunction with the previously described modification to produce the watermark perturbation on op. Using this method, we can generate modifications that, when embedded into the hardware, produce the desired watermark signature under $\mathbf{x}_k$.

V. EXPERIMENTAL VERIFICATION

A. Evaluation Setting

1) Verification Datasets: Our experiments use the Cifar10, Cifar100, and ImageNet (1k) image classification datasets. The Cifar datasets utilize 60 000 32×32 color images organized into 50 000 training and 10 000 testing images. The Cifar10 images are drawn equally from ten classes, while the Cifar100 dataset draws 100 classes. The ImageNet dataset contains 1.2 million high-resolution color images drawn from 1000 classes, including 50 000 validation images. The images were preprocessed to a resolution of 224 according to the evaluated networks' specifications.

TABLE II
GENERAL INFERENCE ON DEEPHARDMARK MODIFIED HARDWARE

Model	Acc%	$T_{ratio}\%$	ΔFid%
VGG11	91.95	0.67	0.206
VGG13	94.03	0.67	0.218
VGG16	93.70	0.75	0.262
VGG19	93.63	0.78	0.234
ResNet34	92.92	0.14	0.009
ResNet50	93.86	0.26	0.009
Dense121	93.30	0.17	0.019

We also use the IMDB and GLUE-SST2 sentiment analysis datasets. The IMDb dataset contains 50 000 text reviews from popular movie titles with a 50/50 train/test split. The Glue-SST2 dataset contains 68 000 training and 1000 validation text reviews from popular movie titles. Both datasets utilize a sentence-level two-way class split.

Code is provided at https://github.com/Jfcleme/Hardware-Watermarks-for-Deep-Learning-Systems.

2) Evaluation DNN Models: For these evaluations, we utilize multiple CNN image classifiers of various depths, including ResNet [46], WideResNet [47], VGG [48], DenseNet [49], and EfficientNet [50]. In addition, we utilize the "large" and "small" 16-patch ViT [51] and Swin transformer [52] as attention-based image classification models. Finally, we use DistilBERT [53] and RoBERTa [54] sentiment analysis models. To remain consistent with other works, we utilize the CNN models provided by the TIMM model library and the transformer models provided by HuggingFace. These models are used as the key DNN or evaluation model, as described in the results. We utilize the standard notations for these models to indicate the specific architectures used in our evaluations.

3) Evaluation Metrics and Hardware Platforms: To evaluate the watermark embedding, we utilize the embedding success rate (ESR), accuracy difference (ΔAcc), fidelity difference (ΔFid), and triggering ratio (T_{ratio}) consistent with the prior work. We test our watermark embedding on the MMU and TinyTPU hardware accelerators. We implement and synthesize these designs in the FPGA and ASIC paradigms using Intel's Quartus and Synopsys Design Compiler, respectively. We defer the detailed description of these materials to DeepHardMark [14].

TABLE III
DEEPHARDMARK FPGA HARDWARE UTILIZATION

	TinyTPU				MMU			
	ALUT	DSP	Reg	Power (mW)	LUT	DSP	Reg	Power (mW)
Watermark-free	4171 (4%)	0 (0%)	1178	528	8486 (7%)	342 (100%)	28575	530
Watermarked	5323 (5%)	0 (0%)	1703	531	8520 (7%)	342 (100%)	28576	530
Overhead	21.6%	0%	44.6%	0.56%	0.41%	0%	$< 0.01\%$	0%

TABLE IV
DEEPHARDMARK ASIC HARDWARE OVERHEAD

	Area	Cells	Power	Time
TinyTPU	0.144%	0.119%	0.169%	0.00%
MMU	0.054%	0.058%	0.039%	0.00%

B. Embedding Results

1) Efficacy Analysis: We first demonstrate the efficacy of embedding hardware watermarks into a deep learning system through the hardware domain. We target a ResNet18 model as the key DNN trained on the Cifar10, Cifar100, and ImageNet datasets for these results. For each key DNN, we randomly select 25 testing images from the corresponding dataset. In this evaluation, we assume the target hardware is the MMU design and targets its ReLU functional blocks. We find a watermark embedding for each key DNN/key sample pair. We developed a framework that allows us to simulate hardware modifications in Pytorch. Using this framework, we observe the binary representation of a model's latent space variables and produce corresponding bit-flips in the binary representations of those values when the modifications are activated.

Using this framework, we calculate ESR, $|\delta_k|$, ΔAcc, and ΔFid. $|\delta_k|\%$ describes the average percentage of target operations in the model that require hardware modification. We also calculate ΔArea, the average percentage area increase in the area of the ASIC design when embedding the modifications. These results provide a general overview of the efficacy and impact of watermark embedding on the deep learning system from both the functional and hardware perspectives. We present these results in Table I. ΔAcc and ΔFid are calculated using 1000 images from the testing dataset excluding the key sample. We utilized several models, seen in Table II, to determine ΔFid as discussed below.

We conclude from these results that the proposed watermark embedding scheme can reliably embed low-impact hardware watermarks into a design. Specifically, we observed an ESR of 100% across these experiments while requiring only a 1% increase in hardware overhead when using the ResNet18 ImageNet classifier as the Key DNN. Furthermore, ΔAcc and ΔFid are under 0.7% for all scenarios while often being significantly lower. From both the hardware overhead and algorithmic functionality, these impacts are minor.

2) Watermark Fidelity Evaluation: This section evaluates the ability of the proposed methodology to embed hardware modifications into a design while preserving the design's algorithmic fidelity. We perform this evaluation by taking the hardware modification sets produced for the Cifar10 ResNet18 classifier generated in the previous experiments and utilizing our simulation framework to alter a broad range of Cifar10 evaluation models. In this setting, we observe the T_{ratio} and ΔFid of each evaluation model under the hardware modification set. The results of this evaluation are presented in Table II.

We can see from these results that in the worst case scenario, ΔFid is only changed by 0.26%, which is minor enough to be considered the effect of optimizations in such hardware designs. However, ΔFid quantifies the accumulation of changes in the model's operations, resulting in altered classifications. In other settings, the T_{ratio} gives us a better insight into fidelity preservation as it indicates how frequently the modifications would introduce minor changes to models. From this perspective, we observe that $<1\%$ of the model's target operations are incorrectly altered. These results indicate that the watermark embedding has a very subtle impact on the hardware's functionality.

3) Hardware Overhead: Here, we evaluate the hardware overhead needed to embed the watermarks determined by DeepHardMark through FPGA and ASIC designs generated in Intell's Quartus and Synopsys Design Compiler, respectively. Targeting the 32×32 MMU hardware design discussed above, we arbitrarily select a Cifar10 ResNet18 modification set generated above and implement a combinational circuit for each of these modifications. We inject this hardware into the Verilog design and synthesize the design for the Cyclone V FPGA. We compare the hardware overhead of this design with the watermark-free version in Table III. We observe that the impact of the modifications is minimal in this setting. For example, there is only a 0.18% increase in the number of LUTs used, while FF and DSP utilization remains unchanged. From the power utilization perspective, we only observe an approximate increase of 0.17%, further verifying the effectiveness of the watermarking scheme.

In addition, we also synthesize ASIC implementations and present the hardware overhead in Table IV for this paradigm. For this perspective, we target both TinyTPU and extend the FPGA MMU design to ASIC. We can translate the watermark modifications to this domain with ease. In this setting, we also observe very little overhead. As seen, the modifications increase the area of the hardware implementation by 0.054% and its power consumption by 0.038%. We present the average increase in area consumption by a random selection of modification sets for each of the ResNet18 models in Table I.

4) Broad Evaluation of DeepHardMark$^+$: We continue our experimental evaluations by presenting the effectiveness of the DeepHardMark$^+$ algorithm. The improvement in computational efficiency of these optimizations allows us to easily extend our results to larger, more complex models and hardware architectures demonstrating a significant benefit.

TABLE V
EVALUATING THE EFFECTIVENESS AND IMPACT OF DEEPHARDMARK$^+$ WATERMARK MODIFICATIONS IN IMAGE CLASSIFICATION

Dataset	Model	Acc%	$ESR\%$	$\lvert\delta_k\rvert\% \pm SD$	ΔAcc% $\pm SD$	ΔFid% $\pm SD$	$T_{ratio} \pm SD$
Cifar10	ResNet18	93.02	100	0.001 ± 0.001	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	ResNet34	93.34	99	0.001 ± 0.001	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Cifar100	ResNet18	87.66	100	0.004 ± 0.003	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	ResNet34	88.54	96	0.003 ± 0.001	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
ImageNet	ResNet18	89.08	92	0.017 ± 0.015	0.11 ± 0.08	0.00 ± 0.00	$4.33 \times 10^{-7} \pm 4.05 \times 10^{-7}$
	ResNet50	92.86	100	0.004 ± 0.009	0.16 ± 0.38	0.00 ± 0.00	$9.10 \times 10^{-8} \pm 2.78 \times 10^{-7}$
	VGG11	87.40	100	0.001 ± 0.002	0.23 ± 0.62	0.06 ± 0.15	$2.90 \times 10^{-8} \pm 7.20 \times 10^{-8}$
	VGG16	96.88	100	0.004 ± 0.010	0.15 ± 0.42	0.06 ± 0.21	$6.30 \times 10^{-8} \pm 1.61 \times 10^{-7}$
	WideResNet50	94.08	97	0.021 ± 0.046	0.13 ± 0.29	0.12 ± 0.28	$4.17 \times 10^{-5} \pm 4.79 \times 10^{-5}$
	EfficientNetB2	95.31	98	0.020 ± 0.048	0.10 ± 0.27	0.10 ± 0.27	$6.80 \times 10^{-8} \pm 9.30 \times 10^{-8}$

TABLE VI
EVALUATING THE EFFECTIVENESS OF DEEPHARDMARK$^+$ IN TRANSFORMERS AND NATURAL LANGUAGE PROCESSING MODELS

Dataset	Model	Acc%	ESR	$\lvert\delta_k\rvert\% \pm SD$
ImageNet	ViT-16-224	97.66	100	0.028 ± 0.036
	ViT-32-224	96.82	96	0.020 ± 0.021
	Swin-b-224	97.34	83	0.004 ± 0.002
IMDb	DistilBERT	92.79	100	0.001 ± 0.001
	RoBERTa	94.66	66	0.001 ± 0.001
GLUE-SST2	DistilBERT	98.85	70	0.001 ± 0.001
	RoBERTa	92.55	70	0.001 ± 0.001

We target a larger hardware design for these experiments containing a 128×128 MMU and show that we can embed watermarks while targeting fewer operations in the key DNN. We utilize the same procedures discussed in Section V-B1.

In Table V, we present the results of our experimental evaluations on the improved methodology on an array of image classifiers, including ResNet50, WideResNet, and EfficientNetB2. We observe that the enhanced algorithm can successfully target these large CNN image classifiers with a high success rate, i.e., >90%. It also produces watermark embeddings that target a smaller percentage of the model's operations than the base algorithm while being less likely to affect the key DNN's accuracy and fidelity negatively. We further highlight the minor impact of the algorithm with the trigger ratio, T_{ratio}, which is very small for all key DNNs.

5) Extension to Alternative Deep Learning Scenarios: In Table VI, we directly extend our experimental evaluations to state-of-the-art transformer-based classifiers and natural language processors. We assume a 128×128 MMU-based hardware design for this evaluation. Similar to the previous experiments, we produce 100 watermark embeddings generated with randomly selected testing inputs targeting the ReLU functional blocks. We utilize our simulation framework to analyze the impact of the watermark embedding on ViT and Swin ImageNet classifiers, as well as DistilBERT and RoBERTa sentiment analysis models. We present the ESR and $|\delta_k|\%$ observed in Table VI. Despite the significant differences between transform models and CNNs, DeepHardMark$^+$ achieves similar levels of performance on ViT and Swin models to their CNN counterparts. We observe a drop in the ESR as we transition to the natural language processing setting. However, the ESR remains above 65% in this setting. Despite this, we note that the embedding DeepHardMark$^+$ produces highly efficient watermark embedding for this domain with less than 0.001% of the model operations being targeted. The high success rates and low impact of these results demonstrate that our methodology can easily extend into other application domains with minor optimizations for the various target scenarios.

6) Hardware Evaluation of DeepHardMark$^+$: We then evaluated the hardware impact of the DeepHardMark$^+$ watermark embedding. For this evaluation, we implement a 32×32 version of the MMU and TinyTPU hardware accelerators in Quartus. We generate a watermark embedding for the ResNet18 Cifar10 classifier and embed modification which produces the watermark in the design. We synthesize the watermarked and watermark-free designs for a Cyclone V FPGA and determine its resource utilization in terms of look-up tables (LUTs), registers (Reg), and digital signal processors (DSPs) as well as its estimated power draw. For the power consumption estimation, we assumed an I/O toggle rate of 12.5%. We present the recorded FPGA utilization in Table VII.

This process is repeated for ASIC design using a Synopsys 32-nm technology node. We utilize the same watermark embedding and hardware designs. For the ASIC design implementations, we determine the area and cell count. Power consumption and propagation delay of the system. The recorded estimations of these characteristics are presented in Table VIII.

From these results, we observe that the impact of the watermark modifications on the hardware overhead is minimal. This finding is consistent with prior evaluations of DeepHardMark. For instance, the increase in register and DSP utilization in the FPGA implementations and the power consumption and propagation delay of the ASIC designs are all less than 0.01%. Furthermore, in the ASIC designs, the largest impact is the increase in the required cells, which requires only a 0.045% expansion. Comparing these results with those presented in Section V-B3, we demonstrate that the DeepHardMark$^+$ algorithm can produce hardware modifications with a decreased impact on the hardware overhead of the target design.

VI. BROADER IMPACT AND FUTURE DIRECTIONS

Some estimates calculate that IC piracy results in as much as $7.5 billion lost in yearly revenue and 11 000 jobs [55].

TABLE VII
HARDWARE UTILIZATION OF DEEPHARDMARK$^+$ IN FPGA DESIGNS

	TinyTPU				MMU			
	LUT	DSP	Reg	Power (mW)	LUT	DSP	Reg	Power (mW)
Watermark-free	4171 (4%)	0 (0%)	1178	528	8486 (7%)	342 (100%)	28575	530
Watermarked	4413 (4%)	0 (0%)	1552	530	8499 (7%)	342 (100%)	28576	530
Overhead	5.8%	0%	31.7%	0.38%	0.15%	0%	$> 0.01\%$	0%

TABLE VIII
HARDWARE OVERHEAD OF DEEPHARDMARK$^+$ IN ASIC DESIGNS

	Area	Cells	Power	Time
TinyTPU	0.042%	0.048%	0.09%	0.01%
MMU	0.013%	0.011%	0.01%	0.04%

Historically, adversaries have been able to pirate enough to build large enough portfolios to mimic major electronics suppliers, such as NEC [56]. In September 2023, an executive employee at SK Hynix, a Korea DRAM supplier, was convicted of pirating counterfeit chips [57]. As the majority of chips are developed through fabless manufacturing, deep-learning hardware accelerators are susceptible to these same supply chains.

Our results demonstrate our methodology is able to reliably and effectively embed watermarks in deep learning accelerators that can be used to identify the hardware. However, the method of verifying the proposed watermark relies on the owner providing the key DNN and samples that activate the hardware watermarks. There is no guarantee that other data will not activate the watermark. Luckily, the low ΔAcc and ΔFid demonstrate that it is very unlikely that an adversary randomly stumbles upon input materials that trigger the watermark. However, it is possible that methodologies to discern or generate the materials needed to activate the watermark exist. To ensure the reliability of such watermarks in real-world scenarios, research efforts should be applied to discovering such methodologies and to defend against them. Such adversarial compromises have been a significant direction of interest from the software perspective. Luckily, the inherent differences between hardware and software make such techniques likely not trivially transferable to this scenario but may provide inspiration for valid points of entry.

Another perspective for improving upon this work is to increase the capacity of the embedded watermarks. As authentication methods continue to develop we anticipate methods for increasing the capacity of embedded watermarks. Powerful methodologies in other domains have allowed for more information to be embedded in the systems, for example, serial numbers, licensing information, and developer identifiers can be embedded in many IPs. Expanding the ability of deep learning authentication beyond the scheme proposed here will be a significant benefit to the field.

VII. CONCLUSION

This work extends the DeepHardMark algorithm, the first watermarking embedding framework for deep-learning hardware, with DeepHardMark$^+$. The extended algorithm improves upon it with a novel methodology of optimizing for the minimal cardinality constraint and an improved intrablock perturbation reduction algorithm. These optimizations allow the algorithms to consistently converge to superior solutions over DeepHardMark while enhancing the algorithm's efficiency. These improvements allow us to expand our experimental evaluations to cover a broader range of key DNNs, including state-of-the-art transformer-based deep learning models. Furthermore, our evaluations targeting the more complex models demonstrate a significant improvement in the performance and impact of the DeepHardMark$^+$ watermarks embeddings over DeepHardMark.

REFERENCES

[1] D. Ghimire, D. Kil, and S.-H. Kim, "A survey on efficient convolutional neural networks and hardware acceleration," *Electronics*, vol. 11, no. 6, p. 945, Mar. 2022.
[2] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.
[3] Y. Hu, Y. Liu, and Z. Liu, "A survey on convolutional neural network accelerators: GPU, FPGA and ASIC," in *Proc. 14th Int. Conf. Comput. Res. Develop. (ICCRD)*, Jan. 2022, pp. 100–107.
[4] D. Zhang et al., "A full-stack search technique for domain optimized deep learning accelerators," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Feb. 2022, pp. 27–42.
[5] S. Boyson, T. M. Corsi, and J.-P. Paraskevas, "Defending digital supply chains: Evidence from a decade-long research program," *Technovation*, vol. 118, Dec. 2022, Art. no. 102380.
[6] F. Koushanfar, "Intellectual property (IP) protection for deep learning and federated learning models," in *Proc. ACM Workshop Inf. Hiding Multimedia Secur.*, Jun. 2022, p. 5.
[7] J. J. Rinsy, N. M. Sivamangai, R. Naveenkumar, A. Napolean, A. Puviarasu, and V. Janani, "Review on logic locking attacks in hardware security," in *Proc. 6th Int. Conf. Devices, Circuits Syst. (ICDCS)*, Apr. 2022, pp. 342–347.
[8] T. Nigussie, J. C. Schabel, S. Lipa, L. McIlrath, R. Patti, and P. Franzon, "Design obfuscation through 3-D split fabrication with smart partitioning," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 9, pp. 1230–1243, Sep. 2022.
[9] D. Sisejkovic, F. Merchant, L. M. Reimann, H. Srivastava, A. Hallawa, and R. Leupers, "Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 3, pp. 1–26, Jul. 2021.
[10] N. N. Anandakumar et al., "Rethinking watermark: Providing proof of IP ownership in modern SoCs," *IACR Cryptol. ePrint Archive*, p. 92, Jan. 2022.
[11] N. M. Jebreel, J. Domingo-Ferrer, D. Sánchez, and A. Blanco-Justicia, "KeyNet: An asymmetric key-style framework for watermarking deep learning models," *Appl. Sci.*, vol. 11, no. 3, p. 999, Jan. 2021.
[12] Y.-Q. Zhang, Y.-R. Jia, X. Wang, Q. Niu, and N.-D. Chen, "DeepTrigger: A watermarking scheme of deep learning models based on chaotic automatic data annotation," *IEEE Access*, vol. 8, pp. 213296–213305, 2020.
[13] M. Shayan, K. Basu, and R. Karri, "Hardware trojans inspired IP watermarks," *IEEE Des. Test.*, vol. 36, no. 6, pp. 72–79, Dec. 2019.
[14] J. Clements and Y. Lao, "DeepHardMark: Toward watermarking neural network hardware," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 4450–4458.

[15] X. Hu, L. Chu, J. Pei, W. Liu, and J. Bian, "Model complexity of deep learning: A survey," *Knowl. Inf. Syst.*, vol. 63, pp. 2585–2619, Aug. 2021.

[16] L. Bernstein, A. Sludds, R. Hamerly, V. Sze, J. Emer, and D. Englund, "Freely scalable and reconfigurable optical hardware for deep learning," *Sci. Rep.*, vol. 11, no. 1, pp. 1–12, Feb. 2021.

[17] G. Li et al., "Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning," *J. Syst. Archit.*, vol. 124, Mar. 2022, Art. no. 102431.

[18] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks," *Synth. Lect. Comput. Archit.*, vol. 15, no. 2, pp. 1–341, 2020.

[19] C. Gao, T. Delbruck, and S.-C. Liu, "Spartus: A 9.4 TOp/s FPGA-based LSTM accelerator exploiting spatio-temporal sparsity," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 1, pp. 1098–1112, Jan. 2024.

[20] A. R. Díaz-Rizo, H. Aboushady, and Haralampos-G. Stratigopoulos, "Leaking wireless ICs via hardware trojan-infected synchronization," *IEEE Trans. Depend. Secure Comput.*, vol. 20, no. 5, pp. 3845–3859, Sep. 2023.

[21] A. Jain, Z. Zhou, and U. Guin, "Survey of recent developments for hardware trojan detection," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[22] X. Hu et al., "Practical attacks on deep neural networks by memory trojaning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 6, pp. 1230–1243, Jun. 2021.

[23] J. Clements and Y. Lao, "Hardware trojan design on neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[24] P. Kadian, S. M. Arora, and N. Arora, "Robust digital watermarking techniques for copyright protection of digital data: A survey," *Wireless Pers. Commun.*, vol. 118, no. 4, pp. 3225–3249, Jun. 2021.

[25] J. Jia, Y. Wu, A. Li, S. Ma, and Y. Liu, "Subnetwork-lossless robust watermarking for hostile theft attacks in deep transfer learning models," *IEEE Trans. Depend. Secure Comput.*, early access, Jul. 28, 2022, doi: 10.1109/TDSC.2022.3194704.

[26] B. Zhao and Y. Lao, "Class-oriented poisoning attack," in *Proc. Winter Conf. Appl. Comput. Vis.*, 2022, pp. 3741–3750.

[27] Y. Wu, M. Xue, D. Gu, Y. Zhang, and W. Liu, "Sample-specific backdoor based active intellectual property protection for deep neural networks," in *Proc. IEEE 4th Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2022, pp. 316–319.

[28] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *Proc. 27th USENIX Conf. Security Symp.*, Baltimore, MD, USA, 2018, pp. 1615–1631.

[29] M. Xue, S. Sun, Y. Zhang, J. Wang, and W. Liu, "Active intellectual property protection for deep neural networks through stealthy backdoor and users' identities authentication," *Appl. Intell.*, vol. 52, pp. 16497–16511, Mar. 2022.

[30] Y. Li, Z. Zhang, J. Bai, B. Wu, Y. Jiang, and S.-T. Xia, "Open-sourced dataset protection via backdoor watermarking," 2020, *arXiv:2010.05821*.

[31] Y. Lao, W. Zhao, P. Yang, and P. Li, "DeepAuth: A DNN authentication framework by model-unique and fragile signature embedding," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 9595–9603.

[32] X. Cao, J. Jia, and N. Z. Gong, "IPGuard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, May 2021, pp. 14–25.

[33] A. Chakraborty, A. Mondai, and A. Srivastava, "Hardware-assisted intellectual property protection of deep learning models," in *Proc. 57th ACM/IEEE Des. Automat. Conf.*, Jul. 2020, pp. 1–6.

[34] A. Viticchié et al., "Assessment of source code obfuscation techniques," in *Proc. IEEE 16th Int. Work. Conf. Source Code Anal. Manipulation (SCAM)*, Oct. 2016, pp. 11–20.

[35] I. Khairunisa and H. Kabetta, "PHP source code protection using layout obfuscation and AES-256 encryption algorithm," in *Proc. 6th Int. Workshop Big Data Inf. Secur. (IWBIS)*, Oct. 2021, pp. 133–138.

[36] A. Sengupta and S. P. Mohanty, "Functional obfuscation of DSP cores using robust logic locking and encryption," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2018, pp. 709–713.

[37] H. M. Kamali, K. Z. Azar, F. Farahmandi, and M. M. Tehranipoor, "Advances in logic locking: Past, present, and prospects," *IACR Cryptol. ePrint Arch.*, p. 260, Jan. 2022.

[38] J. Rajendran, O. Sinanoglu, and R. Karri, "Regaining trust in VLSI design: Design-for-trust techniques," *Proc. IEEE*, vol. 102, no. 8, pp. 1266–1282, Jul. 2014.

[39] A. L. Oliveira, "Robust techniques for watermarking sequential circuit designs," in *Proc. 36th Conf. Design Autom.*, M. J. Irwin, Ed. New York, NY, USA: ACM, 1999, pp. 837–842.

[40] I. Torunoglu and E. Charbon, "Watermarking-based copyright protection of sequential functions," *IEEE J. Solid-State Circuits*, vol. 35, no. 3, pp. 434–440, Mar. 2000.

[41] M. Lewandowski and S. Katkoori, "Enhancing PRESENT-80 and substitution-permutation network cipher security with dynamic 'Keyed' permutation networks," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2021, pp. 350–355.

[42] K. Dzhanashia and O. Evsutin, "Low complexity template-based watermarking with neural networks and various embedding templates," *Comput. Electr. Eng.*, vol. 102, Sep. 2022, Art. no. 108194.

[43] B. Wu and B. Ghanem, "$]\ell_p$-box ADMM: A versatile framework for integer programming," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 7, pp. 1695–1708, Jul. 2019.

[44] X. Dong et al., "GreedyFool: Distortion-aware sparse adversarial attack," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 11226–11236.

[45] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 25–34.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[47] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Brit. Mach. Vis. Conf.*, R. C. Wilson, E. R. Hancock, and W. A. P. Smith, Eds. Durham, U.K.: BMVA Press, 2016.

[48] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, Y. Bengio and Y. LeCun, Eds., 2015.

[49] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 4700–4708.

[50] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[51] A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent.*, 2021.

[52] Z. Liu et al., "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 10012–10022.

[53] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," 2019, *arXiv:1910.01108*.

[54] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.

[55] J. Blake. (Feb. 2022). *Planning, Policy, Production and Piracy Converging Economic and Technology Drivers of the Global Semiconductor Shortage*. Accessed: Nov. 24, 2023. [Online]. Available: https://www.linkedin.com/pulse/planning-policy-production-piracy-converging-economic-jeffrey-blake/?trk=articles_directory

[56] P. Clarke. (May 2006). *Fake NEC Company Found, Says Report*. Accessed: Nov. 24, 2023. [Online]. Available: https://www.eetimes.com/fake-nec-company-found-says-report/

[57] J. Eun-Soo. (Sep. 2023). *Supplier's Employees Found Guilty of Pirating SK Hynix Chip Technology to China*. Accessed: Nov. 24, 2023. [Online]. Available: https://koreajoongangdaily.joins.com/news/2023-09-14/business/industry/1869545