DeepAttest: An End-to-End Attestation Framework for Deep Neural Networks

Huili Chen
University of California, San Diego
huc044@ucsd.edu

Cheng Fu
University of California, San Diego
cfu@ucsd.edu

Bita Darvish Rouhani
University of California, San Diego
Microsoft
bita.rouhani@microsoft.com

Jishen Zhao
University of California, San Diego
jzhao@ucsd.edu

Farinaz Koushanfar
University of California, San Diego
farinaz@ucsd.edu

ABSTRACT

Emerging hardware architectures for Deep Neural Networks (DNNs) are being commercialized and considered as the hardware-level *Intellectual Property* (IP) of the device providers. However, these intelligent devices might be abused and such vulnerability has not been identified. The unregulated usage of intelligent platforms and the lack of hardware-bounded IP protection impair the commercial advantage of the device provider and prohibit reliable technology transfer. Our goal is to design a systematic methodology that provides *hardware-level IP protection* and *usage control* for DNN applications on various platforms. To address the IP concern, we present *DeepAttest*, the first on-device DNN attestation method that certifies the legitimacy of the DNN program mapped to the device. DeepAttest works by designing a device-specific *fingerprint* which is encoded in the weights of the DNN deployed on the target platform. The embedded fingerprint (FP) is later extracted with the support of the Trusted Execution Environment (TEE). The existence of the pre-defined FP is used as the attestation criterion to determine whether the queried DNN is authenticated. Our attestation framework ensures that only authorized DNN programs yield the matching FP and are allowed for inference on the target device. DeepAttest provisions the device provider with a practical solution to limit the application usage of her manufactured hardware and prevents unauthorized or tampered DNNs from execution.

We take an Algorithm/Software/Hardware co-design approach to optimize DeepAttest's overhead in terms of latency and energy consumption. To facilitate the deployment, we provide a high-level API of DeepAttest that can be seamlessly integrated into existing deep learning frameworks and TEEs for hardware-level IP protection and usage control. Extensive experiments corroborate the fidelity, reliability, security, and efficiency of DeepAttest on various DNN benchmarks and TEE-supported platforms.

ISCA '19, June 22–26, 2019, Phoenix, AZ, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6669-4/19/06…$15.00
https://doi.org/10.1145/3307650.3322251

CCS CONCEPTS

• **Security and privacy → Authentication**; **Security in hardware**; • **Computing methodologies → Machine learning**.

KEYWORDS

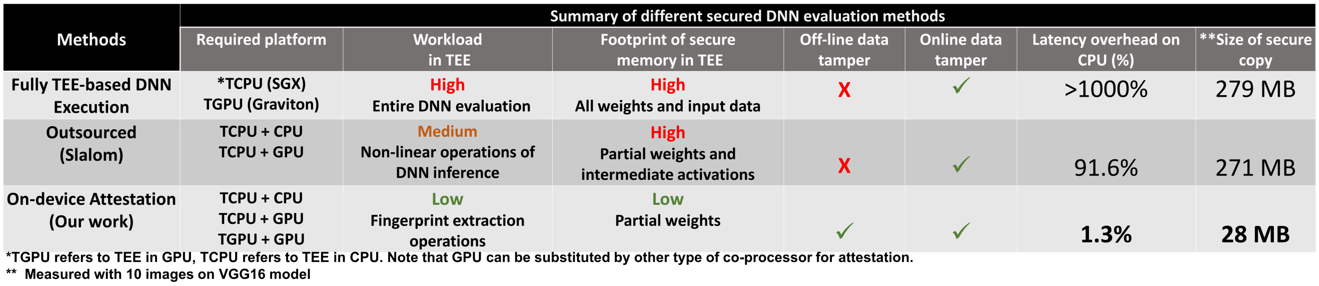
IP Protection, Deep Neural Networks, Software/Hardware Co-design, Attestation

ACM Reference Format:
Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. 2019. DeepAttest: An End-to-End Attestation Framework for Deep Neural Networks. In *The 46th Annual International Symposium on Computer Architecture (ISCA '19), June 22–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3307650.3322251

1 INTRODUCTION

Deep Neural Networks (DNNs) are increasingly adopted in various fields ranging from biomedical diagnosis, nuclear engineering to computer vision and natural language processing due to their unprecedented performance [10, 14]. Methodological and architecture-level advancements have been proposed to improve the performance and efficiency of DNN training/execution on diverse platforms [8, 18, 41]. While the distribution of intelligent devices facilitates the deployment of DNNs in real-world settings, IP concerns may arise in the supply chain. The customers might misuse the device for illegal or unauthorized DNN applications. In this paper, we are motivated to provide hardware-level IP protection and usage control via on-device DNN attestation to protect the commercial advantages of the device providers.

Prior works have identified the IP concern in the deployment of current Deep Learning (DL) models. Various DNN watermarking techniques have been proposed to prevent copyright infringement of soft neural IP [7, 12, 48]. The watermark is embedded in the distribution of weights/activations [12, 48], or the decision boundary [1, 12]. Existing DNN watermarking provides DNN ownership proof in the *software/functionality* level while the authentication overhead and the underlying computing platform are not taken into account. Developing an efficient and effective on-device DNN attestation methodology is challenging since the attestation scheme is required to: (i) *Preserve the performance (e.g., accuracy)* of the deployed DNN; (ii) *Provide reliable and secure attestation decision*; (iii) *Incur low latency and power consumption* to ensure applicability in real-time DNN applications and resource-constrained systems.



Methods	Summary of different secured DNN evaluation methods						
	Required platform	Workload in TEE	Footprint of secure memory in TEE	Off-line data tamper	Online data tamper	Latency overhead on CPU (%)	**Size of secure copy
Fully TEE-based DNN Execution	*TCPU (SGX) TGPU (Graviton)	High Entire DNN evaluation	High All weights and input data	X	✓	>1000%	279 MB
Outsourced (Slalom)	TCPU + CPU TCPU + GPU	Medium Non-linear operations of DNN inference	High Partial weights and intermediate activations	X	✓	91.6%	271 MB
On-device Attestation (Our work)	TCPU + CPU TCPU + GPU TGPU + GPU	Low Fingerprint extraction operations	Low Partial weights	✓	✓	**1.3%**	**28 MB**

*TGPU refers to TEE in GPU, TCPU refers to TEE in CPU. Note that GPU can be substituted by other type of co-processor for attestation.
** Measured with 10 images on VGG16 model

Figure 1: Comparison of existing secure DNN techniques and our work.

We develop an end-to-end on-device attestation framework called *DeepAttest* to address the above challenges. DeepAttest, for the first time, extends IP protection to hardware/device-level leveraging the support from the Trusted Execution Environment (TEE). DeepAttest takes the pre-trained model and security parameters from the device provider as its inputs, provisioning a trade-off between security level and attestation overhead. A set of verifiable, functionality-preserved DNNs are returned. The marked models pass the customized attestation and execute inference on the pertinent device. DeepAttest effectively detects malicious modifications and prevents illegitimate models from execution.

DeepAttest framework consists of two key phases: (i) Marking stage (off-line): DeepAttest generates a device-specific signature (fingerprint) associated with each target hardware for the platform provider and embeds it in the probabilistic distribution of selected parameters within the deployed DNN. (ii) Attestation stage (online): DeepAttest utilizes a hybrid trigger mechanism to prevent static and dynamic data tamper. When the attestation is activated, DeepAttest securely extracts the fingerprint (FP) of the deployed DL model with TEE's support and compares it with the true value stored in the secure memory. The queried DNN is decided to be legitimate and permitted for normal inference if it yields a matching FP. Otherwise, the DNN program fails the attestation and its execution is aborted. By introducing DeepAttest, this paper makes the following contributions:

- **Enabling effective on-device attestation for DNN applications.** The proposed end-to-end attestation framework is capable of verifying the legitimacy of an unknown DNN with high reliability (preventing unauthenticated DNNs from execution) and high integrity (allowing legitimate DNNs to execute normal inference).
- **Characterizing the criteria for a practical attestation in the domain of deep learning.** We introduce a comprehensive set of metrics to profile the performance of pending DNN attestation techniques. The introduced metrics allow DeepAttest to provide a trade-off between the security level and the attestation overhead.
- **Leveraging an Algorithm/Software/Hardware co-design approach to devise an efficient attestation solution.** Our device-aware framework is equipped with careful design optimization to ensure minimal overhead and enhanced security. As such, our solution provides a lightweight on-device attestation scheme that can be applied to resource-constrained systems.
- **Investigating DeepAttest's performance on various DNN benchmarks and TEE-supported platforms.** We perform extensive experiments on DNNs with different topologies using TEE-supported CPU (Intel SGX) and GPU (via simulation) platforms.

DeepAttest opens a new axis for the growing research in secure DL. Our approach is orthogonal to existing secure DL methods that aim to verify the correctness of DNN execution or preserve the privacy of sensitive data. DeepAttest paves the way for on-device attestation and platform-aware usage control for DNN applications.

2 BACKGROUND AND MOTIVATION

2.1 Secure DNN Evaluation on Hardware

TEE Protection Mechanism. Modern CPU hardware architectures provide TEEs to ensure secure execution of confidential applications using program isolation. Intel SGX [52], ARM TrustZone [31] and Sanctum [11] are examples of TEEs TEEs are called *enclaves* in SGX. To prevent malicious programs from interfering executions in TEE, data is encrypted by Memory Encryption Engine (MEE) before its is put into the Enclave Page Cache (EPC) located in the Processor Reserved Memory (PRM). We refer to this process as *secure memory copy.* Programs inside the TEE can read or write the data outside of the TEE while the programs outside of the TEE is not allowed to access the EPC. TEEs on other platforms utilize similar mechanisms to isolate the execution of the protected program by securing memory access to the code and data of the confidential program. Besides the CPU-level TEE support, Graviton [49] proposes an GPU architecture design to provide TEEs.

Comparison between Secure DNN Techniques. Figure 1 illustrates the comparison between the state-of-the-art secure DNN techniques and DeepAttest in terms of platform requirement, incurred workload in TEE, resistance to off-line/online data tamper, and capability of verifying DNN inference results. A quantitative overhead comparison is given in the last two columns. Detailed explanations about the of the overhead are given in Section 6.6. Existing secure DNN inference can be divided into two categories: full execution inside the TEE, and outsourcing partial computations from the TEE to untrusted environments. *DeepAttest identifies a new security dimension named 'device-level' IP protection and usage control.* Note that DeepAttest is orthogonal to the techniques that provide verifiable results and privacy-preserving property. We discuss the limitation of the present secure DNN techniques below.

▪ **Fully TEE-based DNN Evaluation.** A naive way to ensure trusted DNN inference is to run all computations inside the TEE. However, such an approach incurs a prohibitive overhead due to: (i) Limited PRM size in TEEs for execution. For instance, the enclave memory size of Intel SGX is 128MB, which is much smaller than the

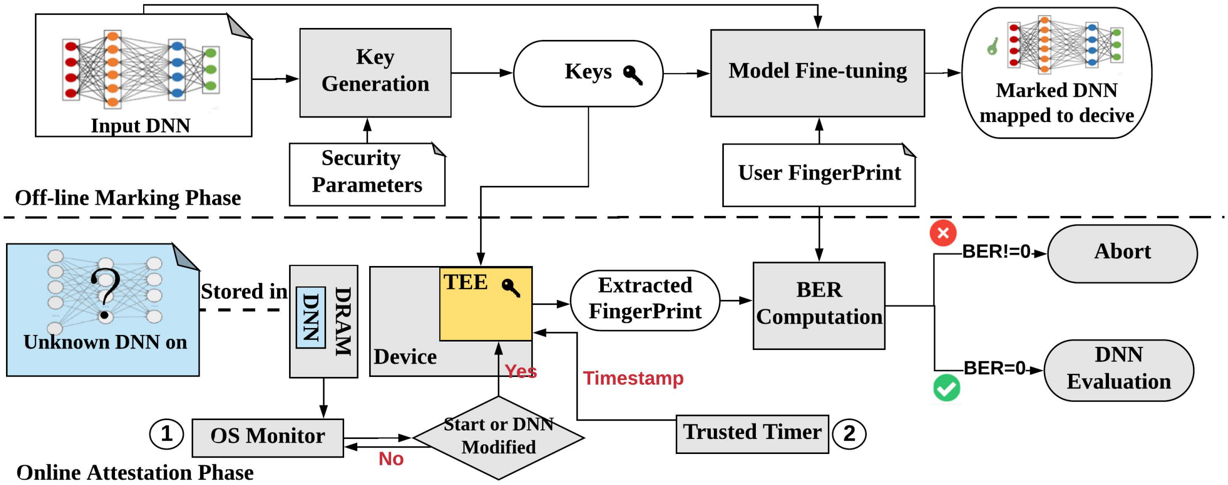


Figure 2: DeepAttest's global flow for on-device DNN attestation. In the off-line marking stage, the device manufacture obtains her secret FP keys and a set of marked DL models. The FP keys are then stored in the secure memory of the TEE on the target device. The user is required to purchase the marked DNN from the device provider to pass the online attestation and execute normal inference. Deployment of unauthorized DNN programs and malicious fault injection will be detected.

parameter size of a contemporary DL model (e.g., 189MB for ResNet-101). As such, the weights need to be reloaded for different inputs. (ii) Encryption and decryption by MEE. Data that is communicated with the secure memory needs to be encrypted/decrypted; (iii) Additional hardware behaviors due to CPU's context switch [19]. Operations such as flushing TLBs and out-of-order execution pipeline are necessary for security consideration [11], incurring extra overhead.

▪ **Outsource-based Secure DNN Inference.** Slalom [47] is a framework for secure DNN execution on trusted hardware that guarantees integrity. It partitions DNN computations into non-linear and linear operations. These two parts are then assigned to the TEE and the untrusted environment for execution, respectively. Freivalds' algorithm [35] is used to verify the integrity of linear computations performed on the untrusted GPU. Slalom reduces the overhead compared to fully TEE-based inference. However, intermediate results need to be transferred into TEE to complete DNN forward propagation, incurring large communication overhead.

▪ **TEE-based Attestation (Our Work).** Unlike the previous methods, DeepAttest is the first framework that can prevent both off-line and online data tamper. More specifically, Slalom and fully TEE-based DNN evaluation can guarantee result integrity and protect the weight parameters from online data tamper. However, these two methods are vulnerable to off-line data tamper (e.g., fault injection) where the attacker modifies the data stored in the untrusted memory before it is used in the secure DNN inference.

2.2 Motivation

Prior works have focused on model ownership proof using software-level DNN watermarking [12, 48]. Existing watermarking techniques are oblivious of the deployed platform and verification overhead, thus the security and efficiency of their execution on the hardware are not guaranteed (detailed in Section 7). DeepAttest is motivated to address the above deficiencies. It provides attestation-based IP protection that is bounded to hardware and restricts device usage for DNN applications. We identify three challenges of developing a practical on-device DNN attestation method below.

(C1) Functionality-preserving. The attestation scheme shall not degrade the performance of the original DL model. Since authenticated DNNs are allowed for normal inference, their functionality (e.g., accuracy) shall be preserved to provide the desired service.

(C2) Security and Reliability. On-device attestation shall be secure and yield reliable decisions under a strong threat model. Note that the attack surface of device-level attestation is larger than the one of software-level attestation.

(C3) Low Overhead. The attestation protocol shall incur negligible overhead to ensure its applicability in real-time data applications and resource-constrained systems.

The constraint *C1* imposes the algorithm/software-level challenge on the design. Challenges *C2* and *C3* need to be resolved from algorithm/software/hardware all three levels. We explicitly develop systematic design principles to tackle the identified challenges C1-C3 as detailed in Section 3.

3 DEEPATTEST OVERVIEW

DeepAttest is the first DNN attestation framework for device IP protection/usage control and is applicable to any computing platform with TEE support. Figure 2 illustrates the global flow of DeepAttest. The target *device* can be used with a co-processor (e.g., ASIC, FPGA), in which case usage control can be extended to it. On-device attestation can be performed on the co-processor if it has TEE support. In this section, we present the *Design Principles* (DPs) of DeepAttest to address the corresponding challenges in Section 2.2.

(DP1) Regularization-based DNN Fingerprinting. To preserve the functionality of the pertinent DNN program (C1), DeepAttest explores the over-parameterization of high dimensional DNNs and utilizes regularization to encode the device-specific FP in the DL model. Regularization [3, 40] is a common approach to alleviate model over-fitting [39, 44]. We detail the two key phases of DeepAttest's algorithm/software design below.

Table 1: Requirements for an effective on-device attestation technique of deep neural networks.

Requirements	Description
Fidelity	Functionality of the deployed DNN shall not be degrade as a result of FP embedding in the marking stage.
Reliability	Online attestation shall be able to prevent unauthorized DNN programs (including full-DNN program substitution and malicious fault injection) from executing on the specific device.
Integrity	Legitimate DNN programs shall yield the matching FP with high probability and run normal evaluation.
Efficiency	The online attestation shall yield negligible overhead in terms of latency and energy consumption.
Security	The attestation method shall be secure against potential attacks including fault injection and FP forgery.
Scalability	The attestation technique shall be able to verify DNNs of varying sizes.
Generalizability	The DNN attestation framework shall be compatible with various computing platforms.

▪ **Off-line DNN Marking.** DeepAttest takes the pre-trained DL model and the owner-defined[1] security parameters as its inputs. DeepAttest then outputs the FP secret keys along with the corresponding set of marked DNNs that are ready-to-be-deployed on the target device. Note that FP embedding is a one-time task performed by the owner before the authorized models are deployed on the target device. Furthermore, the secret FP keys stored in the secure memory can be updated after device distribution. This is feasible since current TEEs typically support remote attestation (RA) that allows secure memory update of the TEE. Details about off-line DNN marking are given in Section 4.1.

▪ **Online DNN Attestation.** DeepAttest utilizes a hybrid triggering scheme where a TEE-based attestation process is instantiated when the static or the dynamic trigger is activated. During the attestation phase, the marked weights data that carries the FP is copied to the secure memory inside the TEE. The FP is then extracted from the weights within the TEE. Finally, the Bit Error Rate (BER) between the recovered FP and the ground-truth one is computed. The verified DNN with zero BER is allowed to run normal inference. Illegitimate DL models with non-zero BERs are aborted. Details about the attestation protocol are discussed in Section 4.2.

(DP2) TEE-based Attestation. Hardware-bounded attestation has larger attack surface compared to the one in software-level [32]. The program might be corrupted by the adversary in an untrusted execution environment. As such, the computation involved in attestation shall be performed securely. DeepAttest utilizes TEE-supported trusted hardware to guarantee the security and reliability of the attestation result (addressing C2, detailed in Section 4.2).

(DP3) Algorithm/Software/Hardware Co-design. We present multiple design optimization techniques to enhance the efficiency and security of DeepAttest. As a result, our framework is applicable to real-time data applications and resource-constrained systems. DeepAttest's hardware optimization includes: (i) Data pipeline that hides the majority of the TEE latency during attestation; (ii) Early termination that avoids unnecessary computation; (iii) Shuffled data storage that provides stronger security against fault injection. These optimization address challenge C3 as detailed in Section 5.

3.1 DNN Attestation Metrics

We introduce a comprehensive set of criteria to evaluate the performance of a DNN attestation technique. Table 1 details the criteria for an effective DNN attestation methodology. *Fidelity* requires that the functionality (e.g., accuracy) of the pre-trained model shall not be degraded after the off-line DNN marking. *Reliability* and *integrity* means that the attestation approach shall prevent unauthorized DNNs from executing (low false alarm rate of FP detection) and allow normal inference of legitimate DNNs (high detection rate of the embedded FP), respectively. A reliable attestation method is also desired to satisfy the *security* requirement such that the attestation decision is trustworthy. *Efficiency* requires that the overhead (i.e., latency, power consumption) incurred by attestation shall be negligible. *Scalability* and *generalizability* ensure that the attestation method can be applied to DNNs of various size and diverse TEE-supported hardware devices, respectively. DeepAttest satisfies all the requirements listed in Table 1 as shown in Section 6.

3.2 Assumptions and Threat Model

DeepAttest's Assumptions. We aim to design a robust attestation scheme that yields reliable decisions in various situations. More specifically, we consider the following three adversarial levels: (i) Operating System (OS) is trusted and can lock the pages allocated for the weight data. In this case, the weights in the main memory will not be tampered or evicted. (ii) OS cannot lock the memory but is able to provide information about the pages associated with the weights (e.g., page fault or page modified); (iii) Hardware provides a trusted timestamp while OS might be corrupted (does not satisfy the requirements in (i) and (ii)). DeepAttest tackles with the above scenarios by designing a hybrid trigger from two sources, OS and the secure timer, as detailed in Section 4.2.

Threat Model. Adversaries might try to bypass the on-device attestation. We detail three potential attacks below and demonstrate the experimental results of DeepAttest's security in Section 6.

(i) Full-DNN Program Substitution. Untrusted users may attempt to misuse the distributed device by mapping illegitimate DL model to it. In this case, the adversary is assumed to know the physical address of the deployed DNN (e.g., by eavesdropping) and substitutes the original content with his target unauthorized DNN. DeepAttest is motivated to address the susceptibility of the intelligent device to such attacks.

(ii) Fingerprint Forgery Attack. In order to successfully pass the attestation, the attacker might attempt to forge the device-specific signature stored in the secure memory inside the TEE. More specifically, the adversary may use brute-force searching and try to find the exact secret key used in FP embedding to reconstruct the device's fingerprint and yield zero BER.

(iii) Fault Injection. Besides the program-level replacement, the attacker may also conduct more fine-grained memory content modification attacks. We assume a strong attack model where the adversary might know the memory allocation on the target device and randomly selects memory blocks for malicious purpose (e.g., malware, pirating sensitive information). Such local memory modification is stealthier than the DNN-program substitution discussed above and poses potential threats to the intelligent device.

[1] We use owner and device provider interchangeably in the paper.

4 DEEPATTEST DESIGN

DL models typically feature non-convex loss functions with many local minima that are likely to yield similar accuracy [9]. DeepAttest takes advantage of the non-uniqueness of non-convex problems to embed the device's FP in the distribution of the selected weights. The embedded FP is later extracted in the attestation phase as the identifier to determine the legitimacy of the DNN. We use Convolution Neural Networks (CNNs) and Residual Networks to illustrate DeepAttest's workflow. Note that DeepAttest is generic and can be applied to other network architectures. We detail the two key stages discussed in Section 3 below.

4.1 Off-line DNN Marking

Algorithm 1 outlines the steps involved in DeepAttest's off-line DNN marking (i.e., FP embedding) for one intermediate layer. The extension to multi-layer fingerprinting is straightforward. DeepAttest's DNN marking consists of the following three steps:

```
ALGORITHM 1: Fingerprint embedding for one hidden layer.
INPUT: Pre-trained unmarked DNN ($\mathcal{T}$); Training data
    ($\{X^{train}, Y^{train}\}$); Location of the target layer ($l$) with
    embedding dimension ($N$); Code length ($v$); Resilience
    level ($k$); Embedding strength ($\gamma$).
OUTPUT: A set of marked DNNs ($\{\mathcal{T}_1^*, ..., \mathcal{T}_b^*\}$); FP keys.
❶ Key Generation:
        $C_{v\times b} \leftarrow Construct\_Codebook\ (v, k, 1)$
        $U_{v\times v} \leftarrow Generate\_Basis\_Matrix\ (v)$
        $X_{v\times N} \leftarrow Generate\_Projection\_Matrix\ (v, N)$
❷ Fingerprint Construction:
        $F_{v\times b} \leftarrow Construct\_Fingerprints\ (C, U)$
❸ Model Fine-tuning: For each user $j$ ($j = 1, ..., b$), train the DNN
    on $\{X^{train}, Y^{train}\}$ with the corresponding FP-specific loss:
        $\mathcal{L} = \mathcal{L}_0 + \gamma \cdot Mean\_Square\_Error(\mathbf{f_j} - X\mathbf{w_j})$.
Return: Marked DNNs ($\{\mathcal{T}_1^*, ..., \mathcal{T}_b^*\}$), FP keys
    ($l$, $C_{v\times b}$, $U_{v\times v}$, $X_{v\times N}$).
```

❶ **Key Generation.** Besides the position of the target layer that carries the FP, DeepAttest's FP keys consist of three components: a codebook C, an orthogonal basis matrix U, and a projection matrix X. We explain the design of each component as follows:

(i) Devices Codebook: Given the code length v and the maximal number of supported users b specified by the owner, DeepAttest generates a codebook $C \in \mathbb{B}^{v\times b}$ for the device provider. The codebook is randomly generated where each column of C is a unique code-vector associated with a specific device. The code-vector is stored in the secure memory within the TEE on the target hardware.

(ii) Orthogonal Basis Matrix: The orthogonal matrix $U_{v\times v}$ is generated from element-wise Gaussian distribution for security consideration [51]. The columns of U are used as the basis vectors for FP construction in step 2.

(iii) Projection Matrix: The owner's secret projection matrix $X_{v\times N}$ is generated from standard normal distribution $\mathcal{N}(0, 1)$ where N is the embedding dimension of the target layer. We explicitly illustrate the design of FP carrier for convolutional (conv) layers and fully-connected (FC) layers below:

■ **Convolutional Layer.** The weight matrix of a conv layer is a 4D tensor $W \in \mathbb{R}^{D\times D\times F\times H}$ where D is the kernel size, F and H is the number of input and output channels, respectively. We average the weight W over the output channel dimension and stretch the result to a vector $\mathbf{w}$. The FP is embedded in the projection of the vector $\mathbf{w} \in \mathbb{R}^N$ (detailed in step 3) where the embedding dimension is $N = D \times D \times F$.

■ **Fully-Connected Layer.** The weights of the FC layer is a 2D matrix $W_{F\times H}$ where F is the input dimension and H is the number of units. Similar to the processing for conv layers, we average W over the last dimension and use the resulting vector $\mathbf{w} \in \mathbb{R}^N$ to carry the FP. Note that the embedding dimension $N = F$ here.

❷ **Fingerprint Construction.** DeepAttest constructs code modulated FPs as follows. Given the codebook $C_{v\times b}$ obtained in step 1, the coefficient matrix $B_{v\times b}$ for FPs is computed from the linear mapping $b_{ij} = 2c_{ij} - 1$ where $c_{ij} \in \{0, 1\}$. For j^{th} user, the corresponding FP is crafted as the linear combination of basis vectors in U (obtained in step 1) with $\mathbf{b_j} \in \{\pm 1\}^v$ as the coefficient vector:

$$\mathbf{f_j} = \sum_{i=1}^{v} b_{ij}\mathbf{u_i}, \tag{1}$$

❸ **Model Fine-tuning.** The FP designed from Equation (1) is embedded in the weight parameters of the selected layer in the pre-trained model by incorporating the FP-specific embedding loss to the conventional loss function ($\mathcal{L}_0$):

$$\mathcal{L} = \mathcal{L}_0 + \gamma\ Mean\_Square\_Error(\mathbf{f} - X\mathbf{w}). \tag{2}$$

Here, γ is the embedding strength that controls the contribution of the additive FP embedding loss $\mathcal{L}_{FP} = Mean\_Sqaure\_Error(\mathbf{f} - X\mathbf{w})$. The vector $\mathbf{w}$ is the flattened averaged weights of the target layers that carry the FP information. DeepAttest minimizes $\mathcal{L}_{FP}$ together with the conventional loss during DNN training to enforce the FP constraint in the distribution of weights in the selected layer.

4.2 Online DNN Attestation

The secret FP keys generated from the off-line marking stage are stored in the secure memory inside the TEE. The returned DL models that carry the device-specific FPs are deployed on the target platform and stored in the untrusted memory for later execution. Algorithm 2 outlines the two main steps in the online attestation stage. It takes the FP keys and the weights in the marked layers as the inputs. The BER between the extracted FP from the queried weights and the ground-truth FP (included in FP keys) is returned as the output. We detail each step as follows.

1 **Hybrid Attestation Trigger.** DeepAttest leverages a *hybrid trigger* mechanism for activating DNN attestation as shown in Figure 2. A *static* trigger signal is generated when the OS detects that a DNN program requests to start. it enables DeepAttest to prevent off-line data tamper if the attacker tries to modify the memory content stored in untrusted environment. During program execution, the *dynamic* trigger is generated from two sources: (i) Memory change signal provided by OS monitoring. OS keeps monitoring the status change of pages allocated for the DNN program and raises a dynamic trigger signal if any online data pages modification is detected; (ii) A timestamp signal from the trusted timer [23]. The dynamic trigger from the secure timestamp has a fixed interval and provides enhanced security when the OS memory monitoring signal is tampered. Incorporating the dynamic triggering scheme is important to enable online data tamper detection. The final trigger signal is the logic-OR of the static and the dynamic signal. DeepAttest is able to detect both off-line and online data tamper due to the incorporation of the static and dynamic trigger, respectively.

2 **Secure Fingerprint Detection.** When the trigger is enabled, the queried DNN program is suspended until the attestation finishes. The fingerprint of the DL model is extracted with the TEE support and compared with the true value stored in the secure memory. More specifically, DeepAttest first acquires the weights of the marked layer by moving them from the untrusted memory into the secure memory within the TEE. Meanwhile, OS locks the pages allocated for the queried DNN program if it has the capability. The core of the online attestation is recovering the code-vector $\mathbf{c}' \in \{0,1\}^v$ from the weight data W. This reconstruction involves matrix multiplication and an element-wise hard-thresholding as shown in Algorithm 2. Note that the recovering of each bit in $\mathbf{c}'$ is independent, DeepAttest leverages this observation from two perspectives (detailed in Section 5.2): (i) *Data pipeline*: DeepAttest utilizes data independence for parallel computation, thus reduces the attestation latency; (ii) *Scalability*: DeepAttest allows transferring partitioned blocks of the weight data into the secure memory in TEE. As such, DeepAttest attestation can be applied to arbitrary large DL model and TEE platforms with limited secure memory.

Algorithm 2 Fingerprint extraction in the attestation stage.

```
INPUT: Queried DNN ($\mathscr{T}'$); Decoding threshold ($\tau$);
    Owner's FP keys ($\{l, C, U, X\}$).
OUTPUT: Computed $BER$ of fingerprint matching.
1  Trigger Generation: Produce a hybrid trigger signal:
       $S_{hybrid} \leftarrow S_{static} \vee S_{dynamic}$
2  if $S_{hybrid}$ == True then
       Acquire weights in the marked layer:
           $\mathbf{w}' \leftarrow Get\_Marked\_Weights\ (\mathscr{T}', l)$
       Extract the FP vector: $\mathbf{f}' \leftarrow X\mathbf{w}'$
       Recover the coefficient vector: $\mathbf{b}' \leftarrow \mathbf{f}'^T U$
       Decode the code-vector:
           $\mathbf{c}' \leftarrow Hard\_Thresholding\ (\mathbf{b}', \tau)$
       Check FP matching:
           $BER \leftarrow Compute\_BER(\mathbf{c}', \mathbf{c})$
       Return: Obtained $BER$ for FP matching.
3: else
       Go back to step 1
```

■ **Attestation Case Study:** We demonstrate how DeepAttest authenticates a DNN program using the extracted FP below. Let us consider a codebook $C_{7\times7}$ shown in Equation (3). The FPs for 7 users shown in Equation (4) are constructed using the columns of the codebook C and the basis matrix U as described in Section 4.1.

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \tag{3}$$

$$\begin{cases} \mathbf{f}_1 = -\mathbf{u}_1 - \mathbf{u}_2 + \mathbf{u}_3 - \mathbf{u}_4 + \mathbf{u}_5 + \mathbf{u}_6 + \mathbf{u}_7, \\ \cdots \\ \mathbf{f}_6 = +\mathbf{u}_1 + \mathbf{u}_2 - \mathbf{u}_3 - \mathbf{u}_4 + \mathbf{u}_5 + \mathbf{u}_6 - \mathbf{u}_7, \\ \mathbf{f}_7 = +\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3 - \mathbf{u}_4 - \mathbf{u}_5 - \mathbf{u}_6 + \mathbf{u}_7, \end{cases} \tag{4}$$

Let us take the first device FP as an instance where the ground-truth FP ($\mathbf{f}_1$) is stored in the secure memory. For a legitimate DNN program whose weights carry the FP vector $\mathbf{f}' = \mathbf{f}_1$, the corresponding coefficient vector can be recovered by computing the correlation with the basis vectors:

$$\mathbf{b}' = \mathbf{f}'^T[\mathbf{u}_1, ..., \mathbf{u}_7] = [-1, -1, +1, -1, +1, +1, +1].$$

The code-vector is then extracted by the inverse linear mapping $c_{ij} = \frac{1}{2}(b_{ij} + 1)$, resulting in $\mathbf{c}' = [0, 0, 1, 0, 1, 1, 1]$. Since the recovered code-vector $\mathbf{c}'$ exactly matches $\mathbf{c}_1$ (first column of the codebook in Eq. (3)), DeepAttest returns $BER = 0$ and allows the queried DNN program to execute normal inference. This example shows that DeepAttest respects the *integrity* requirement in Table 1 can effectively detect the embedded FP in the legitimate DNN. Furthermore, the computation required to recover the FP code-vector is simple, rendering DeepAttest lightweight.

Note that for the first device, any DNN program that cannot yield a matching code-vector (thus non-zero BER) will be aborted by our attestation protocol. As such, DeepAttest also satisfies *reliability* requirement by terminating unauthorized DNN programs. The secret FP keys are stored in the secure memory inside the TEE and is tamper-resistant, suggesting DeepAttest's *security*.

5 DEEPATTEST OPTIMIZATION

DeepAttest framework integrates innovative hardware optimization techniques to ensure high security (Section 5.1) and efficiency (Section 5.2). We explicitly discuss two design optimization below.

5.1 Shredder Storage

DeepAttest utilizes a *'shredder'* storage format instead of continuous storage to provide stronger security against the fault injection attack. More specifically, DeepAttest shuffles the weights and stores the resulting data in the untrusted memory. Note that the shuffling pattern is determined by the owner in the off-line stage, thus the attacker has no knowledge about the locations of the marked weights. This method is intrigued by the idea of Oblivious RAM (ORAM) where the memory blocks are duplicated and shuffled to hide the memory access pattern from adversary [16]. However, we consider a different scenario where data shuffling is performed inside the model parameters to prevent fault injection.

Figure 3 illustrates the intuition of higher security using shredder storage. When the marked weights are stored continuously, the adversary can easily find a safe position to inject malicious memory blocks without overlapping with the marked region (shadowed area). If the blocks containing the marked weights are shuffled, our online attestation scheme is more likely to yield a non-zero BER and abort the program.

DeepAttest enforces a theoretical upper bound on the success rate (η) of the attacker who aims to perform fault injection while ensuring BER=0 in the attestation stage. We formulate the mathematical problem as follows. Assuming all weight parameters of the deployed DNN takes N memory blocks where n of them carry the

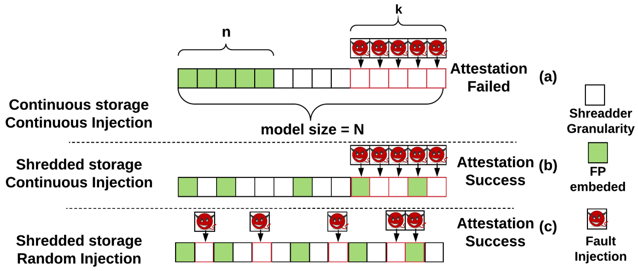


Figure 3: Security optimization using shredder storage

device-specific FP information. The attacker tries to inject k segments of equal size (s blocks) into the memory while still be able to pass the online attestation. We assume the attacker does not know the storing pattern of weights and randomly inserts his malicious blocks into the memory. One can see that the intervals between the adjacent marked blocks determine the success probability of the attacker. DeepAttest's shredder storage independently and randomly allocates each marked block, thus the distribution of the locations of the marked blocks can be modeled as a *Poisson Process* with the rate $\lambda = \frac{n}{N}$. As such, the interval X between the neighboring two marked blocks is a random variable with the distribution function:

$$P(X \geq s) = e^{-\lambda s}. \quad (5)$$

For n marked blocks, the corresponding interval sequence has length $n+1$, thus can be denoted as $S_X = \{X_1, X_2, ..., X_{n+1}\}$. Recall that X_i are i.i.d and satisfies exponential distribution parameterized by λ. To successfully insert a single segment, the adversary can only select the intervals which have values equal or larger than the segment size s. The number of such intervals satisfying the constraints $\{X_i \in S_X, X_i \geq s\}$ is a random variable with *Binomial Distribution*. More specifically, B is a binomial distributed random variable where $n+1$ independent trials are conducted with the success rate $p = P(X \geq s)$ given in Equation (5) for each trial:

$$P(B = b) = C_{n+1}^{b} p^{b} (1-p)^{n+1-b}. \quad (6)$$

Combining the above analysis, the success rate of attacker (parameterized by s and k) can be computed as:

$$P_a = \sum_{b=k}^{n+1} P(B = b) \cdot \frac{C_b^k}{C_{n+1}^k}. \quad (7)$$

DeepAttest provides a tunable security level against fault injection by selecting the parameter λ and the upper bound on the attack success rate $P_a < \eta$. Figure 4 illustrates the detection performance of DeepAttest's shredder storage. The x-axis is the injection ratio defined as $\phi = \frac{k \cdot s}{N}$ and the y-axis is the marked ratio $\lambda = \frac{n}{N}$. Increasing the injection ratio or the marked ratio results in lower attack success rate. Figure 4 suggests tne trade-off between the resistance against fault injection (tolerated injection ratio) and the DeepAttest's overhead (marked ratio for fingerprinting).

5.2 Efficient Attestation

■ **Customized Attestation Interval.** As described in Section 4.2, DeepAttest utilizes both static and dynamic trigger signals to activate the attestation. Such a hybrid triggering mechanism provides a trade-off between security and efficiency. Intuitively, checking the FP with a smaller interval gives stronger security while incurring larger overhead. The device provider can leverage this trade-off to customize the configuration of the attestation trigger in her device based on her resource budget and the desired security level.

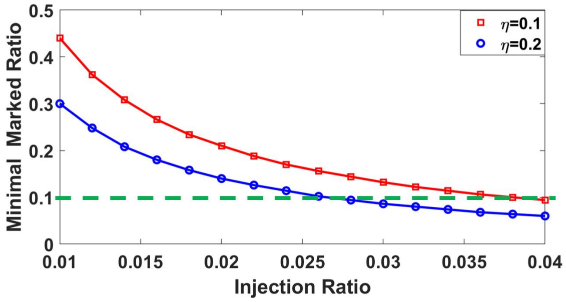


Figure 4: DeepAttest's detection performance of fault injection. The relation between the injection ratio and the minimal marked ratio with varying attack success rate is shown.

■ **Data Pipeline** Due to the limited size of enclave memory, we pipeline secure memory copy and the FP computation in TEE as shown in Figure 5. To this end, we create two pipelined TEE threads to move the partitioned weight data into the TEE and extract the FP, respectively. The enclave memory occupied by FP extraction is freed once the computation is finished and no intermediate results need to be stored. As such, the weight parameters of large sizes can be easily fitted into the enclave memory. Note that our pipeline optimization is feasible since the reconstruction of each bit in the FP is *independent* and *parallelizable* as discussed in Section 4.2. Such a data partitioning scheme further improves DeepAttest's *scalability*.

■ **Early Termination.** To further reduce the attestation overhead, we avoid unnecessary computation and communication using early termination. More specifically, the online attestation terminates and yields the abortion command once a mismatch between the extracted FP segment and the pre-specified device-specific FP is detected as shown in Figure 5.

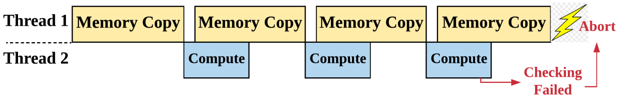


Figure 5: Illustration of DeepAttest's data pipeline and early termination for TEE-based attestation.

6 EVALUATIONS

We assess the performance of DeepAttest according to the requirements discussed in Table 1. A codebook $C_{31\times31}$ that accommodates 31 users is used in our experiments. Without explicit hyperparameter tuning, we set the embedding strength to $\gamma = 0.1$ and fine-tune pre-trained DNN for 5 epochs with the learning rate in the last stage for off-line DNN marking. The threshold for code-vector extraction is set to $\tau = 0.85$. We investigate DeepAttest's performance on Intel-SGX (TEE-support CPU platform) and Graviton-based TEE simulation (GPU platform) [49]. DeepAttest is orthogonal to the existing secure DNN evaluation techniques shown in Figure 1 and we provide a horizontal overhead comparison in Section 6.6. Details about the hardware platforms and DNN benchmarks are discussed below.

Experimental Setup. To evaluate DeepAttest on TEE-supported CPUs, we use SCONE [2], a secure container with Intel SGX [52] execution support. When the hybrid trigger is activated, DeepAttest instantiates an attestation process as an enclave inside the SGX engine. We use a host desktop with a i7-7700k processor and measure the energy consumption using *pcm-monitor* utility.

For the evaluation on trusted GPU, we build a TEE simulator for GPUs based on the architecture design proposed in Graviton [49] since there are no existing TEE-supported GPUs available. GPU can use the device driver to monitor the state of pages inside its memory [49], thus providing DeepAttest with the trigger signal for attestation. Our TEE-supported GPU simulator restricts the size of the secure memory to 300MB using memory partition technique. To ensure isolation, our simulator performs encryption and decryption on the data interacting with the secure memory. More specifically, the weight data stored in the untrusted DRAM is first encrypted by our GPU simulator using authenticated encryption (AES in GCM mode) and copied to the secure memory [49]. The resulting data is then decrypted before it is used in TEE-based FP extraction in the online attestation stage. The encryption and decryption latency follows the [45]. We use Nvidia RTX 2080 as the GPU base and measure the power consumption using *nvidia-smi* utility.

DNN Benchmarks Summary. We corroborate DeepAttest's effectiveness on various DNN benchmarks and summarize them in Table 2. Since DeepAttest utilizes a hybrid trigger whose overall activation interval is uncertain, we assume the average trigger interval is $f = 100$, meaning that the attestation is run once every 100 images. We emphasize that DeepAttest enables the owner to customize the trigger configuration and show the attestation overhead under different intervals in Section 6.5.3. We set the minimal marked ratio to $\lambda_m = 0.1$, ensuring a maximal success rate of fault injection $\eta = 0.1$ under injection ratio $\phi > 0.04$ (Figure 4).

Table 2: Evaluated benchmark summary.

Benchmark	Dataset	Model Size (MB)	Multiply-Add Operations (Mops)	Marked Layer Size (MB)
MNIST-CNN	MNIST [29]	1.3	24	0.13 (10.1%)
CIFAR-WRN	CIFAR10 [28]	2.4	198	0.29 (12.3%)
VGG16	ImageNet [13]	276.7	25180	28.3 (10.2%)
MobileNet	ImageNet [13]	8.4	569	1.05 (12.6%)

DeepAttest API. Our end-to-end solution provides a highly-optimized API compatible with current DL frameworks and can perform the two key phases in Algorithm 1 and 2. Furthermore, DeepAttest API provisions *tunable security level and attestation overhead* by allowing the owner to specify the security parameters including the TEE platform (CPU/GPU/other co-processors) for attestation, upper bound on fault injection success η, tolerated injection ratio ϕ, code-vector length v, and the trigger configuration.

6.1 Fidelity

Table 3 shows the test accuracy of the baseline model and the corresponding marked model for each benchmark in Table 2. The marked accuracy is the average value of the total 31 fingerprinted models. One can see that the accuracy of the marked model is comparable to the one of the baseline model, indicating that DeepAttest's off-line marking phase preserves the functionality of the pre-trained model. Slight accuracy improvement can be observed in several benchmarks. This is due to the fact that adding regularization to the training process helps to mitigate model over-fitting [12, 44].

6.2 Reliability and Integrity

■ **Reliability.** The *reliability* criterion requires that the unauthenticated DNN program shall not be allowed for execution, which is equivalent to yielding a non-zero BER for the queried model in the online attestation stage. We consider the following two sources of an illegitimate DNN: (i) Arbitrary unmarked DNN programs. The malicious user may intend to overuse the device by executing a DL model that is not authorized by the owner; (ii) Fault injection into an authenticated DNN program. The adversary may perform fault injection on the legitimate DNN program for malicious purpose (e.g.,malware insertion). Note that the first scenario can be considered as a special case of the second one where the level of faulty injection is sufficiently large.

To evaluate DeepAttest's reliability under the above unintended modifications, we add random Gaussian noise with zero mean, different standard deviation (magnitude of noise) and different spatial range (percent of modified elements) to the weight matrix in the marked layer. Figure 6 shows the resulting BER of the extracted FP after adding noise to the weights in the marked conv layer and the marked FC layer. One can see that the extracted BER becomes non-zero for small values of noise range and noise magnitude, indicating that DeepAttest can effectively forbid the maliciously modified DNN program from execution.

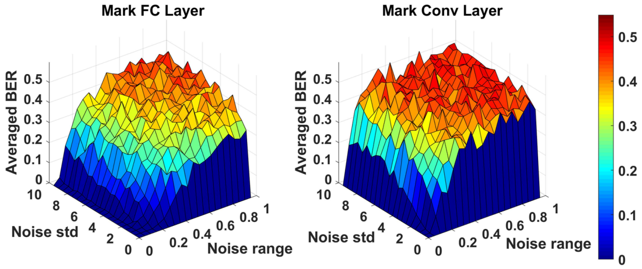


Figure 6: Reliability assessment under noise.

■ **Integrity.** The *integrity* criterion means that the legitimate DNN program shall be able to pass the attestation and execute normal inference. Such a requirement suggests that the attestation protocol shall yield a high FP detection rate (BER=0) for marked models. Figure 6 indicates that DeepAttest respects the integrity criterion since the BER is zero when no noise is added to the marked weights.

6.3 Security

DeepAttest is secure against fingerprint forgery attack. To construct a DNN program that has the same device FP as the one stored in the secure memory inside the TEE, the adversary needs to know: (i) The DNN marking method (i.e., Algorithm 1); (ii) The secret projection matrix X, orthogonal matrix U, and the code-vector $\mathbf{c}$; (iii) Memory addresses of the marked weights stored in shredder storage format. Using brute force search to find all the above information is prohibitively expensive. DeepAttest might be compromised if the underlying TEE is attacked. For instance, Intel SGX has been identified to be vulnerable to side-channel attacks [6, 30]. To address the susceptibility, hardware- and software-based defenses have been proposed [30, 33]. DeepAttest is orthogonal to these methods and can be further secured when integrated with them.

6.4 Qualitative Overhead Analysis

We provide a qualitative analysis of the DeepAttest's overhead. Since the DNN marking is an off-line, one-time process, we focus on the overhead in the online attestation phase here. Recall that the

Table 3: Fidelity requirement. The baseline accuracy is preserved after fingerprint embedding in the underlying benchmarks.

Benchmark	**MNIST-CNN** [25]		**CIFAR-WRN** [48]		**VGG16** [42]		**MobileNet** [20]	
Setting	Baseline	Marked	Baseline	Marked	Baseline	Marked	Baseline	Marked
Test Accuracy (%)	99.52	99.66	91.85	92.03	91.20	91.23	85.83	85.75

weights in the marked layers are transferred from the untrusted memory to the secure memory inside the TEE to extract the FP as outlined in Algorithm 2. The data communication overhead is $O(NH)$ where N is the embedding dimension and H is the number of output channels/units as described in Section 4.2. To reduce attestation computation overhead, DeepAttest *pre-computes* the product X^TU used in FP extraction ($b \leftarrow \mathbf{w}^T \cdot X^TU$ in Algorithm 2). As such, the computation complexity of online attestation is $O(vN)$. The above overhead analysis holds for both conv and FC layers.

6.5 Efficiency

We use the latency and energy consumption per image on the untrusted CPU/GPU as the base value and measure the relative overhead of DeepAttest. As shown in Figure 7, DeepAttest incurs on average 7.2% and 4.4% relative latency overhead on the TEE-support CPU and GPU platforms across all benchmarks, respectively. The average energy overhead of DeepAttest is 4.1% and 1.2% for CPU and GPU devices. The normalized energy overhead incurred by DeepAttest is low since the power in the attestation is much smaller compared to one of DNN inference on a given platform. The normalized overhead of DeepAttest depends on the DNN architecture for both TEE-supported CPU and GPU platforms. More specifically, DeepAttest incurs smaller latency and energy overhead on DL models with larger size (parameter count) and more operations. This is due to the fact that small DNNs (e.g., MNIST-CNN) have lower base overhead compared to large models (e.g., VGG16). Comparing the overhead on different platforms, DeepAttest is more efficient when executed in the TEE in GPUs than CPUs.

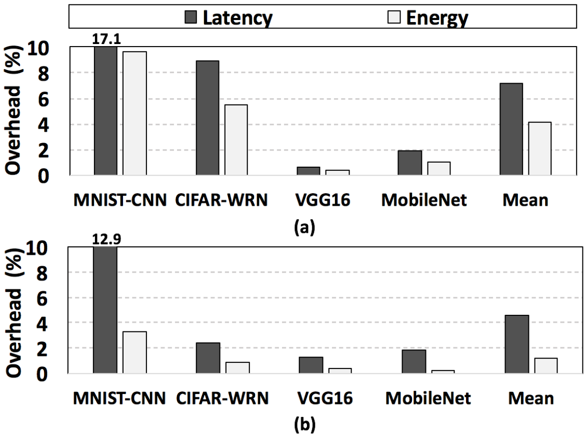


Figure 7: DeepAttest's normalized latency and energy overhead on TEE-supported (a) CPU and (b) GPU platforms.

6.5.1 *Overhead Breakdown.*

To better understand the source and bottleneck of attestation overhead, we analyze the individual runtime of secure memory copy and FP extraction computation. Figure 8 shows the runtime contribution of these two processes. One can see that secure memory copy dominates DeepAttest's overhead in small benchmarks. This is due to the fact secure memory copy involves data loading/encryption and assistant operations executed when entering and exiting an enclave [19]. DeepAttest's secure computation is lightweight since: (i) Secure memory reading is faster than writing [50]; (ii) The involved operations are simple (described in Algorithm 2). The overhead of secure FP computation is affected by the dimensionality of the marked weights, thus varies across different benchmarks as detailed in Section 6.5.3.

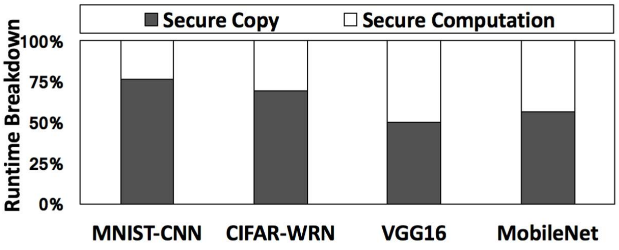


Figure 8: Runtime contribution breakdown of DeepAttest on Intel SGX without dataflow optimization.

6.5.2 *Optimization Improvements.*

We optimize DeepAttest's dataflow using data pipeline as discussed in Section 5.2 to hide the latency of secure FP computation Figure 9 illustrates the effectiveness of data pipeline to reduce the attestation latency. On average, DeepAttest's optimized dataflow engenders 1.42× speedup and further improves its efficiency. Early termination optimization also helps to reduce the overhead. According to Algorithm 2, it is intuitive that the amount of overhead saving benefited from early termination is approximately linear with respect to the position of the first identified FP mismatch segment.

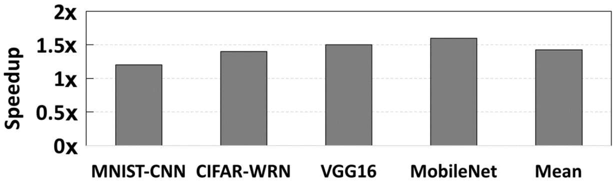


Figure 9: Speedup of DeepAttest's data pipeline optimization for secure FP extraction on Intel SGX.

6.5.3 *Sensitivity Analysis.*

■ **Sensitivity to attestation interval.** DeepAttest leverages a hybrid trigger mechanism to activate the attestation as discussed in Section 4.2. Recall that we denote the average activation interval of the hybrid signal as f (one round of attestation every f inputs). Figure 10 shows DeepAttest's overhead with various attestation interval f on CIFAR-WRN benchmark and Intel-SGX platform. One can see that DeepAttest's overhead decreases linearly when f increases. Higher trigger interval results in smaller normalized attestation overhead for arbitrary DNNs. For instance, the relative latency overhead drops to 1.1% on CIFAR-WRN when $f = 800$.

■ **Sensitivity to the the marked ratio.** The possible values of marked ratio λ for a specific DNN are discrete since DeepAttest performs FP embedding with the granularity of a single layer. Given the owner-specified security level η and the tolerant injection ratio ϕ, DeepAttest finds the minimal marked ratio λ_m (Section 5.1) and the optimal combination of layers that yields the minimal latency.

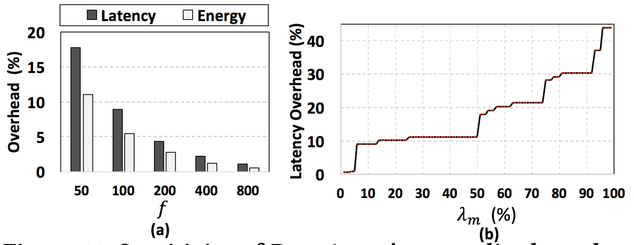


Figure 10: Sensitivity of DeepAttest's normalized overhead to the (a) attestation interval f, (b) minimal marked ratio λ_m on CIFAR-WRN (tested on Intel-SGX).

A large marked ratio λ (i.e., percentage of marked weights) results in a larger latency overhead as shown in Figure 10 (b). Note that λ also impacts the security level of DeepAttest against fault injection as discussion in Section 5.1, thus providing a trade-off between overhead and security.

▪ **Sensitivity to kernel size.** DeepAttest's overhead is affected by the kernel dimension of the marked layer as we analyze in Section 6.5. Figure 10 shows the breakdown of relative runtime overhead for TEE-based attestation and evaluation process as the kernel size changes. For attestation of a conv layer, the runtime overhead is dominated by secure memory copy. However, the runtime discrepancy between secure copy and secure computation becomes smaller as the kernel size increasing since the contribution of the fixed overhead in secure copy (extra hardware operations) is reduced. For TEE-based inference, the runtime of conv kernels is dominated by secure computation. As for FC layers, secure memory copy is the performance bottleneck for both TEE attestation and evaluation due to the large parameter size.

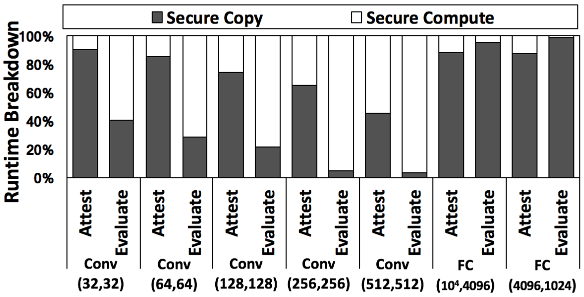


Figure 11: Runtime (relative) breakdown of TEE attestation and evaluation with varying kernel size on Intel-SGX without dataflow optimization. We use conv(F, H) to denote a convolutional layer with size $(3, 3, F, H)$, and FC(F, H) to denote a fuly-connected layer with size (F, H), respectively.

6.6 Comparison with Related Works

In this section, we compared DeepAttest with the state-of-the-art secure DNN evaluation techniques listed in Figure 1. Note that DeepAttest aims to address a new security concern (i.e., hardware-level IP protection and usage control) for DNN applications that has not been identified by previous works. DeepAttest is orthogonal to the existing secure DNN techniques that target at different vulnerabilities of DL models and can be easily integrated within them. As such, we present a horizontal performance comparison to demonstrate the relative overhead required by different security/privacy-protection DNN methods.

In our experiments, we use the open-sourced code of [47] to evaluate the performance of Slalom which aims to verify the integrity of DNN evaluation. Slalom requires quantization of all weights and input data to satisfy their finite-field assumption. We adhere to the quantization technique and the pre-processing method that yields the highest throughput in [47] throughout our experiments. We emphasize that our framework does not require any model compression. As such, our assessment of DeepAttest's overhead is conservative. We use the trusted GPU simulator discussed in Section 6 in the experiments requiring TEE-supported GPU. The design optimization discussed in Section 5 are used. We detail the comparison with related works below.

6.6.1 Comparison of Secure Memory Copy. Figure 12 illustrates the theoretical (minimal) size of secure memory copy required by different secure DNN techniques assuming the TEE is not memory-bounded. Slalom [47] incurs large overhead of secure memory copy since it outsources linear operations of DNN inference to the untrusted GPU. Therefore, all intermediate activations need to be transferred into the TEE to complete non-linear operations. This results in an approximately linear secure copy size with respect to the number of evaluated images as shown in Figure 12 (a) and (b). Fully TEE-based DNN evaluation only requires to transfer all weight data and input data, thus is less sensitive to the number of inputs. DeepAttest's memory copy size is not sensitive to the number of inputs since it adopts a hybrid triggering scheme where the attestation is performed every batch of f images. Furthermore, the secure copy size of DeepAttest is small for a given attestation interval due to the deployment of shredder storage optimization, which ensures security for a smaller value of the marked ratio λ.

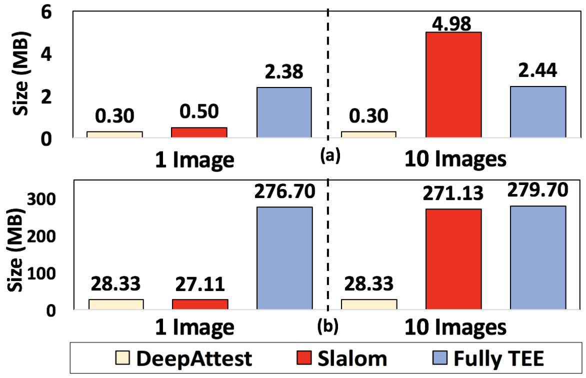


Figure 12: Comparison of the theoretical secure memory copy size to the TEE required by different secure DNN techniques on (a) CIFAR-WRN and (b) VGG16 benchmark.

6.6.2 Comparison of Latency. ▪ **Overhead on CPU-based Inference.** Figure 13 shows the normalized latency required by different secure DNN evaluation methods and DeepAttest where the baseline inference is performed on the untrusted CPU. Implementing DNN inference fully inside TEE is on average 12.34× slower than the baseline evaluation on the untrusted CPU. Slalom [47] outsources linear operations to the untrusted CPU and non-linear parts to Intel-SGX, resulting in an average normalized latency of 1.72× to provide verifiable results. DeepAttest incurs negligible relative latency of 0.7% and 1.9% on VGG16 and MobileNet respectively, thus is highly efficient.

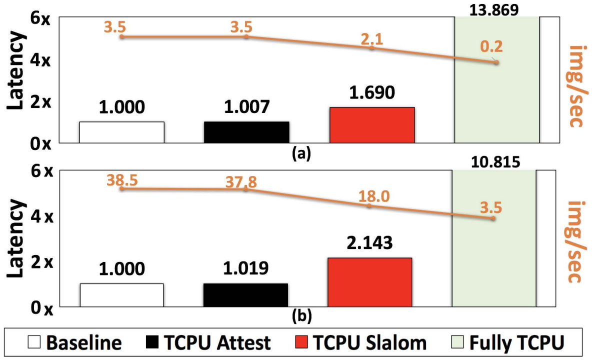


Figure 13: Comparison of relative latency between different secure DNN techniques when the inference is run on CPU. VGG16 (a) and MobileNet (b) are evaluated.

■ **Overhead on GPU-based Inference:** Figure 14 shows the normalized overhead of different secure DNN methods and DeepAttest where the baseline inference is performed on the untrusted GPU. Full DNN inference in the TEE-supported GPU results in an average normalized latency of 8.75× due to the overhead of isolated execution and secure memory access. In this setting, Slalom [47] outsources the linear operations to the untrusted GPU and computes the nonlinear part in Intel-SGX (TCPU), resulting in a normalized latency of 6.43× and 5.69× on VGG16 and MobileNet, respectively.

DeepAttest can perform the FP extraction computation either in the trusted CPU or the trusted GPU if the TEE exists on the pertinent GPU. More specifically, DeepAttest results in 19.1% and 15.7% additional latency when attesting VGG16 and MobileNet on the TEE-supported CPU (Intel-SGX), respectively. Alternatively, DeepAttest can attest the target DNN program using the TEE support inside the GPU to avoid data communication between CPU and GPU. In this case, DeepAttest incurs only 1.3% and 1.8% extra latency on VGG16 and MobileNet, respectively.

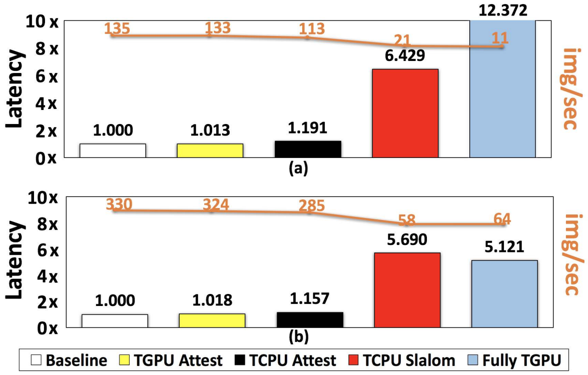


Figure 14: Comparison of normalized latency incurred by different secure DNN methods when the inference is run on GPU. VGG16 (a) and MobileNet (b) are assessed here.

7 RELATED WORK

■ **DNN Watermarking.** A line of research has focused on addressing the soft-IP concern of DL models using digital watermarking [1, 12, 34, 48]. The authors in [48] encode the WM in the transformation of weights by adding constraints to the original objective function. [1, 34] extend DNN watermarking to remote cloud service. They craft specific image-label pairs and embed them in the decision boundary of the model. [12] presents the first data-aware watermarking approach by embedding the WM in the activations.

All of the above mentioned DNN watermarking techniques target at *software-level model authorship proof.* Note that a naive implementation of DNN watermarking on the hardware device is insufficient to provide an efficient and trustworthy attestation solution due to the unawareness of resource management and potential attacks. As such, these methods are not suitable for hardware-level IP protection. [1, 34] require DNN inference of multiple inputs on the local device and TEE-supported WM checking, which is prohibitively costly. Compared to weight-based watermarking [48], DeepAttest's FP extraction involves fewer computations since no extra sigmoid function is required. In this paper, we develop an efficient on-device attestation scheme that ensures the legitimacy of deployed DNN with negligible overhead.

■ **Trusted Execution Environment.** Previous research [4, 46] has paved the path for secure isolated execution on general purpose processors. Intel SGX [52] is the most widely used TEE with user interface. The vulnerabilities of SGX to potential attacks such as spectre[26], cache[17] or other side-channel attacks are later identified. Besides TEE for CPU platforms, a growing amount of research has been done to provide TEE for other hardware platforms. For instance, TyTan [5] and TrustLite [27] are TEEs proposed for embedded systems. DeepAttest is generic and can be extended to these computing platforms with TEE support.

■ **Privacy-Preserving DNN.** Beyond integrity and data tamper, privacy is another critical concern in the DL domain. Various techniques have been suggested for privacy-preserving (PP) DNN inference and training [21, 22, 36, 37]. CryptoNet [15] leverages Homomorphic Encryption (HE) to achieve PP-inference with prohibitive latency due to extensive computations. Gazelle [24] accelerates the LHE-based inference by exploiting SIMD operations. Garbled Circuit (GC) is an alternative approach [38, 43] to HE which has smaller computation overhead but requires more communication.

8 CONCLUSION

In this paper, we develop a systematic solution to provide device-level IP protection and usage control for DNN applications. We propose DeepAttest, the first on-device DNN attestation framework that verifies the legitimacy of the deployed DNN before allowing it to execute normal inference. DeepAttest leverages an Algorithm/Software/Hardware co-design principle and incorporates various design optimization techniques to minimize the overhead. Our framework allows the device providers to explore the trade-off between security level and atteesation overhead by specifying security parameters including tolerance level of fault injection, marked ratio, and trigger configuration. Extensive experimental results corroborate that DeepAttest satisfies all criteria for a practical attestation scheme including fidelity, reliability, integrity, security, scalability, and efficiency.

ACKNOWLEDGMENTS

This work was supported by ONR under grant number N00014-17-1-2500 and AFOSR MURI under award number FA9550-14-1-0351.

REFERENCES

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. *arXiv preprint* (2018).

[2] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.

[3] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research* 7, Nov (2006), 2399–2434.

[4] Rick Boivie and Peter Williams. 2012. SecureBlue++: CPU support for secure execution. *IBM, IBM Research Division, RC25287 (WAT1205-070)* (2012), 1–9.

[5] F. Brasser, B. El Mahjoub, A. Sadeghi, C. Wachsmann, and P. Koeberl. 2015. TyTAN: Tiny trust anchor for tiny devices. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*.

[6] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: SGX cache attacks are practical. *arXiv* (2017).

[7] Huili Chen, Bita Darvish Rohani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. 2019. DeepMarks: A Secure Fingerprinting Framework for Digital Rights Management of Deep Learning Models. In *ACM International Conference on Multimedia Retrieval (ICMR)*.

[8] Y. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379.

[9] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. 2015. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*.

[10] Ronan Collobert and Jason Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proceedings of the 25th International Conference on Machine Learning*. ACM.

[11] Victor Costan, Ilia Lebedev, Srinivas Devadas, et al. 2017. Secure processors part II: Intel SGX security analysis and MIT sanctum architecture. *Foundations and Trends® in Electronic Design Automation* 11, 3 (2017), 249–361.

[12] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. 2019. DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[14] C Fu, A Di Fulvio, SD Clarke, D Wentzloff, SA Pozzi, and HS Kim. 2018. Artificial neural network algorithms for pulse shape discrimination and recovery of piled-up pulses in organic scintillators. *Annals of Nuclear Energy* 120 (2018), 410–421.

[15] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*. 201–210.

[16] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.

[17] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*. ACM, 2.

[18] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[19] Danny Harnik. 2017. Impressions of Intel SGX performance. https://medium.com/@danny_harnik/impressions-of-intel-sgx-performance-22442093595a.

[20] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint* (2017).

[21] Siam U Hussain and Farinaz Koushanfar. 2019. FASE: FPGA Acceleration of Secure Function Evaluation. In *Field-Programmable Custom Computing Machines*.

[22] Siam U Hussain, Bita Darvish Rouhani, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2018. MAXelerator: FPGA accelerator for privacy preserving multiply-accumulate (MAC) on cloud servers. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[23] Intel. 2017. Intel Software Guard Extensions SDK. https://software.intel.com/en-us/sgx-sdk-dev-reference-sgx-get-trusted-time.

[24] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*. 1651–1669.

[25] Yash Katariya. 2016. MNIST CNN benchmark. https://github.com/yashk2810/MNIST-Keras/tree/master/Notebook.

[26] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre attacks: Exploiting speculative execution. *arXiv* (2018).

[27] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: A Security Architecture for Tiny Embedded Devices. In *Proceedings of the Ninth European Conference on Computer Systems*.

[28] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[29] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86 (1998), 2278–2324.

[30] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Security Symposium, USENIX Security*. 16–18.

[31] ARM LIMITED. 2009. ARM Security Technology - Building a Secure System using TrustZone Technology.

[32] Pieter Maene, Johannes Götzfried, Ruan De Clercq, Tilo Müller, Felix Freiling, and Ingrid Verbauwhede. 2018. Hardware-based trusted computing architectures for isolation and attestation. *IEEE Trans. Comput.* 67, 3 (2018), 361–374.

[33] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. ROTE: Rollback Protection for Trusted Execution. In *26th USENIX Security Symposium (USENIX Security 17)*.

[34] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. 2017. Adversarial frontier stitching for remote neural network watermarking. *arXiv preprint* (2017).

[35] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized algorithms*. Cambridge university press.

[36] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-based Oblivious Deep Neural Network Inference. *USENIX Security* (2019).

[37] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM.

[38] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: Scalable Provably-secure Deep Learning. In *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2:1–2:6.

[39] Bita Darvish Rouhani, Mohammad Samragh, Mojan Javaheripi, Tara Javidi, and Farinaz Koushanfar. 2018. Deepfense: Online accelerated defense against adversarial deep learning. In *Proceedings of the International Conference on Computer-Aided Design*. ACM, 134.

[40] Bernhard Scholkopf and Alexander J Smola. 2001. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

[41] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 764–775.

[42] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv* (2014).

[43] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. 2015. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy*.

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[45] Synopsys. 2017. DesignWare pipelined AES-GCM/CTR core. https://www.synopsys.com/dw/ipdir.php?ds=security-aes-gcm-ctr..

[46] Richard Ta-Min, Lionel Litty, and David Lie. 2006. Splitting interfaces: Making trust between applications and operating systems configurable. In *Proceedings of the 7th symposium on Operating systems design and implementation*.

[47] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. *arXiv* (2018).

[48] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding watermarks into deep neural networks. https://github.com/yu4u/dnn-watermark. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ACM, 269–277.

[49] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 681–696.

[50] Ofir Weisse, Valeria Bertacco, and Todd Austin. 2017. Regaining Lost Cycles with HotCalls: A Fast Interface for SGX Secure Enclaves. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 81–93.

[51] Min Wu, Wade Trappe, Z Jane Wang, and KJ Ray Liu. 2004. Collusion-resistant multimedia fingerprinting: a unified framework. In *Security, Steganography, and Watermarking of Multimedia Contents VI*, Vol. 5306. International Society for Optics and Photonics, 748–760.

[52] Bin Cedric Xing, Mark Shanahan, and Rebekah Leslie-Hurd. 2016. Intel&Reg; Software Guard Extensions (Intel&Reg; SGX) Software Support for Dynamic Memory Allocation Inside an Enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy*. ACM.