Preventing DNN Model IP Theft via Hardware Obfuscation

Brunno F. Goldstein, *Graduate Student Member, IEEE*, Vinay C. Patil, *Graduate Student Member, IEEE*, Victor C. Ferreira, Alexandre S. Nery, *Member, IEEE*, Felipe M. G. França, *Senior Member, IEEE*, and Sandip Kundu, *Fellow, IEEE*

***Abstract*— Training accurate deep learning (DL) models require large amounts of training data, significant work in labeling the data, considerable computing resources, and substantial domain expertise. In short, they are expensive to develop. Hence, protecting these models, which are valuable storehouses of intellectual properties (IP), against model stealing/cloning attacks is of paramount importance. Today's mobile processors feature Neural Processing Units (NPUs) to accelerate the execution of DL models. DL models executing on NPUs are vulnerable to hyperparameter extraction via side-channel attacks and model parameter theft via bus monitoring attacks. This paper presents a novel solution to defend against DL IP theft in NPUs during model distribution and deployment/execution via *lightweight, keyed* model obfuscation scheme. Unauthorized use of such models results in inaccurate classification. In addition, we present an ideal end-to-end deep learning trusted system composed of: 1) model distribution via hardware root-of-trust and public-key cryptography infrastructure (PKI) and 2) model execution via low-latency memory encryption. We demonstrate that our proposed obfuscation solution achieves IP protection objectives without requiring specialized training or sacrificing the model's accuracy. In addition, the proposed obfuscation mechanism preserves the output class distribution while degrading the model's accuracy for unauthorized parties, covering any evidence of a hacked model.**

***Index Terms*— IP protection, model obfuscation, neural processing unit, deep learning.**

I. Introduction

In RECENT years, Deep Neural Networks (DNNs) have attracted considerable attention due to state-of-the-art (SOTA) accuracy in tasks such as image classification, object detection, and natural language processing (NLP) [1]–[3]. A trained DNN model's accuracy hinges on some predicates: 1) A large amount of data, which is the most valuable and private asset, has to be collected and labeled. 2) High-performance computing resources (GPUs, TPUs, and NPUs) are required for days to weeks of training time, including multiple trials. 3) The training process depends on comprehensive field experience to better fine-tune the hyperparameters of the model. Therefore, DNN models are costly to create and serve as valuable intellectual property (IP) storehouses, and it is of utmost importance to defend them against stealing/cloning attacks [4], [5].

While modern mobile devices have built-in neural processing unit (NPU) in processor hardware which is capable of executing highly complex DNN models, the learning process still relies on cloud-based ML providers [6]. Thus, model distribution and deployment present an attractive attack surface. A man-in-the-middle attack can be executed without authentication and encrypted transaction between the edge device and the model providers. Similarly, a man-at-the-end attack can be mounted if the end-device provides no defenses during loading and execution time. Therefore, DNN IPs should encompass an additional security layer tightly coupled to the end-device to thwart unauthorized usage.

SOTA models, such as OpenAI NLP GPT [3], are reaching new heights in numbers of model parameters (≈174 Billion on the GPTv3). With the increasing push for accuracy, the model sizes are growing across the application spectrum. Today's models for mobile edge devices are too large to fit into the neural processing units (NPUs) cache, demanding off-chip DRAM usage. Consequently, creating further opportunities for various attacks such as memory bus monitoring or memory probing [7]. To defend against such attacks, the memory content must be encrypted. Unfortunately, most memory encryption techniques add latency for memory access. Thus, we need a *low latency encryption* solution. Further, memory addresses reveal metadata about memory access patterns, which may be used to infer model hyperparameters and memory activities. Thus, there is a need to *obfuscate* memory access.

This paper proposes deep learning (DL) IP protection solutions for NPU edge devices to deal with the above scenarios. We demonstrate that our proposed solutions achieve IP protection objectives *without requiring individualized training*, a bane found in some previous model obfuscation solutions [8], without sacrificing the model's accuracy, and without significant performance impact. Besides, the proposed

Manuscript received December 14, 2020; revised February 22, 2021; accepted April 16, 2021. Date of publication April 28, 2021; date of current version June 14, 2021. This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior–Brasil (CAPES)–Finance Code 001 and in part by the National Council for Scientific and Technological Development–Brazil (CNPq). This article was recommended by Guest Editor K. Basu. *(Corresponding author: Brunno F. Goldstein.)*

Brunno F. Goldstein, Victor C. Ferreira, and Felipe M. G. França are with the COPPE-System Engineering and Computer Science Program, Federal University of Rio de Janeiro, Rio de Janeiro 21941-914, Brazil (e-mail: bfgoldstein@cos.ufrj.br; vcruz@cos.ufrj.br; felipe@cos.ufrj.br).

Vinay C. Patil and Sandip Kundu are with the Department of Electrical and Computer Engineering, University of Massachusetts Amherst, Amherst, MA 01002 USA (e-mail: vcpatil@umass.edu; kundu@umass.edu).

Alexandre S. Nery is with the Department of Electrical Engineering, Universidade de Brasília (UnB), Brasília 70.910-900, Brazil (e-mail: alexandre.nery@redes.unb.br).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2021.3076151.

Digital Object Identifier 10.1109/JETCAS.2021.3076151

obfuscation mechanism preserves the output class distribution for unauthorized users while degrading the model's accuracy, rendering such users unaware of the defense being deployed. Lastly, the proposed obfuscation technique proves to be a secure layer of IP protection over brute force and genetic algorithm based attacks.

The main contributions of the paper are as follows:

- *Model Obfuscation*. We propose a novel lightweight model obfuscation technique that does not require a specialized training process. We demonstrate that our proposed obfuscation scheme can protect model IP while providing output class distribution obliviousness over three well-known neural networks: AlexNet, ResNet18, and ResNet50. We also validate the security of the proposed obfuscation scheme by performing a brute force and a genetic algorithm based attacks.
- *Secure model distribution*. We envisage an end-to-end deep learning trusted execution system, ranging from secure model distribution via hardware root-of-trust and public-key cryptography (PKC) to a protected edge device deployment/execution via memory encryption of model parameters. We do not claim any novel contribution in this area, but showcase how a complete system can be built with existing technologies to accomplish model obfuscation goals without compromise.

The rest of the paper is structured as follows. Section II briefly reviews DNN and neural model obfuscation background and defines the attack model for IP theft. In Section III and Section IV, we introduce and evaluate the proposed model obfuscation scheme over the IP theft attack model. Section V presents an overview of the envisioned trusted execution system, its threat model, and some discussion on secure model distribution and memory encryption. Section VI concludes the work.

II. Background

This section will briefly address some necessary background on Deep Neural Networks, model obfuscation, and the adversary capabilities to mount a model exfiltration attack.

A. Deep Neural Networks

Deep Convolutional Neural Network (DCNN) is a sequence of connected layers, where each layer is responsible for performing a specific type of computation. Three main layer types compose state-of-the-art DCNNs architectures: *Convolutional* Layer, *Pooling* Layer, and *Fully-Connected* Layer. Convolutional layers hold a set of learnable filters/weights responsible for extracting the low and high-level features from the input data. Non-linear activation functions are computed at the end of each Convolutional layer to make their features more expressive. Pooling layers are responsible for controlling the model overfitting by reducing the input data spatial size. Pooling also impacts the network's required computation cost since the number of parameters decreases accordingly with the layer's depth. Finally, Fully-Connected layers act as a classifier, computing the class scores based on the features extracted by the preceding layers.

DCNNs perform two main phases: *Training* and *Inference*. Weights are first randomly initialized and start being adjusted during the training phase (or learning phase) based on their relevance for each input data/class. As input data flows into the network, an error magnitude is calculated based on the distance between the predicted class and ground-truth. The error is then backpropagated so that each weight can be adjusted based on its local and global relevance. The inference phase only requires the forward step, where the input data flow into the network to predict which class it belongs.

A massive amount of labeled input data, several training iterations that run over weeks, and significant expertise to fine-tune the model hyperparameters are required to achieve a well-trained DCNN model. Therefore, protecting the model structure and parameters/weights is imperative to keep it secure from exfiltration attacks.

B. Neural Model Obfuscation

Neural model obfuscation is an additional security layer to protect deep neural models from structure and parameter piracy. It can be done in different ways, from a key-dependent training process to cryptographic encryption schemes.

The first model obfuscation technique was proposed in [9], where structural obfuscation by *shallow* and sequential networks was employed. This process applies a *joint training* phase using the original network and the obfuscated one as a Teacher-Student approach to achieve an effective structural obfuscation. However, whoever has access to the obfuscated model, even a non-legitimate user, can still run it with no penalties (such as accuracy degradation).

A key-dependent training obfuscation was proposed in [8] where the model is protected by applying an enhanced back-propagation algorithm during the training process. A set of random neuron nodes are *locked* within a key defined by the model provider. Only authorized parties possessing the correct key are capable of *unlocking* the neurons and achieving the correct model execution. This technique not only protects the model during the distribution, but also during the end-device deployment. A hardware root-of-trust is required to unlock the model to only authorized end-users. Nevertheless, this key-dependent training obfuscation approach does not provide scalability since it requires model providers to re-train their entire DCNN model portfolio.

Secure cryptographic techniques can also be employed to protect the model structure and parameters. Model providers would encrypt the model before sending it to end-users. Only authorized parties would be able to decrypt it and retrieve the original model. However, this would incur a significant overhead since the model should be decrypted *on-the-fly* to assure that no side-channel attack would occur when loading it from the main memory to a processing unit.

C. Attack Model

In this work, we consider a man-at-the-end (MATE) attack, where the attacker has full access to an edge device capable of running DCNN models with the support of a Neural Processing Unit (NPU). The DCNN model is developed and

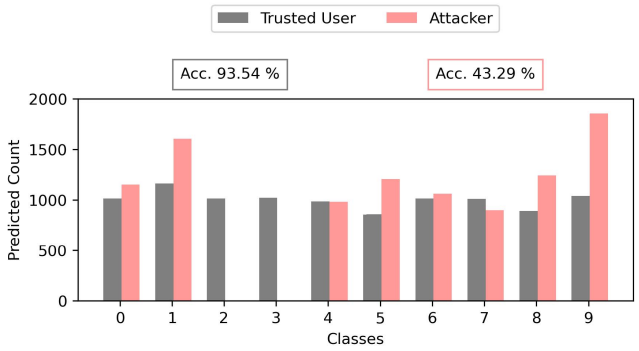

Fig. 1. Output class distribution of HPNN framwork's key-dependent backpropagation [8] using LeNet-4 model over MNIST dataset.

distributed by a Model Provider, which securely distributes its models (more details in Section V) to trusted parties. By reverse-engineering the edge device, the attacker can access the model's topology and weights, but **not** the mapping key (detailed in Sec. III). The attacker's goal is to reuse the neural network model privately or for illegal distribution. In this case, the obfuscation scheme has to defend the model against non-authorized users by decreasing its overall accuracy and preventing the correct model's execution. Moreover, obfuscation has to prevent any information leakage, hiding from the attacker that the current model has been locked.

III. Proposed Model Obfuscation

While training-based techniques provide strong obfuscation of model parameters, it requires a significant amount of time and computational resources, as model providers are required to re-train their entire portfolio of DNN models from scratch, for each user possessing their unique key. Therefore this approach is not scalable.

Further, such a key-dependent backpropagation algorithm [8] strongly disturbs the distribution of outputs, as observed during our experiments and also illustrated in Fig. 1. Change in output distribution may alert the user that the model is compromised. The main goal is to preserve the original output distribution while making the classification as inaccurate as possible for wrong keys.

Our proposed obfuscation relies on the basic structure of deep convolutional neural networks (DCNNs), the convolutional filters, shown in Fig. 2.A. As mentioned in Section II-A, a DCNN is composed of a stack of layers, where each convolutional layer holds a set of filters which are responsible for extracting the low and high-level features from the input data. A filter's parameters results from its local and global relevance for each input data/class provided during the training phase. Therefore, without the correct placement of the filters, the network does not behave accordingly, reducing its overall accuracy. We take advantage of this connectivity to provide a robust and lightweight model obfuscation.

We obfuscate our model considering three possible approaches: *(a)* full filter swaps, *(b)* row/column swaps, and *(c)* hybrid swaps, as shown in Fig. 2.B. In the row/column approach, we randomly select rows or columns from the same filters within the same layer that are then swapped.

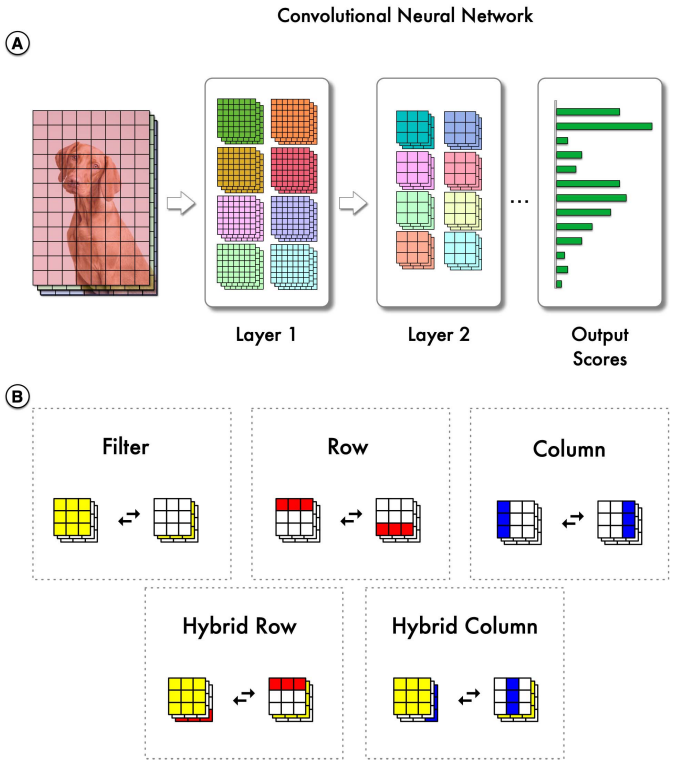

Fig. 2. (A) Convolutional Neural Network layer's structure. (B) Model obfuscation techniques (filter, row, column and hybrid swaps).

The full filters swap considers not only a single row or column, but an entire filter to be swapped. The hybrid approach considers both swap types mentioned above, mixing filters and rows/columns swaps in the same obfuscation process. The obfuscation can achieve high entropy by repeating the swap process for all subsequent layers since the preceding layers' error will be propagated and amplified. The difference between each approach is the impact on the final overall accuracy and the output class distribution. The amount of swaps coupled with the selected structure dictates the obfuscation mode, from stealth to a full defense. The model's accuracy will drop at a small percentage in the stealth mode, but the output class distribution will be preserved, hiding any evidence that the model has been successfully hacked. On the other hand, a full defense obfuscation mode will drastically drop the overall accuracy when under attack. However, after some output class analysis, the attacker might know that the model is locked. It is up to the model provider to choose the obfuscation mode.

Only end-users holding the mapping control key, generated by model providers pre-processing obfuscation phase, can unlock the model's inference. Even if a potential attacker owns the topology and parameters, the model will misbehave, resulting in a performance drop. Furthermore, the proposed swap-based obfuscation avoids re-training the whole model from scratch, enabling fast generation of obfuscated models with a unique mapping key for each end-user. Thus, this solution is not only robust, but also scalable.

A. Implementation Overhead

Model providers are required to generate a mapping key to implement the proposed model obfuscation scheme.

The mapping key is responsible for storing the information of whether a structure should be swapped or not. Therefore, if we considering an n-bit key, one bit will represent one swap, and the total of swaps will be tied to n and the number of convolutional layers in the network (layers share the same key). Additionally, model providers must generate a file containing the information regarding which structure of a layer demands to be swapped. The list of structures (pair of indices) is strongly tied to the mapping key, so only part of the indices' pairs are swapped (bit key equals 1). Therefore, the content of this file can be disclosed with no harm to the obfuscation scheme. After applying the obfuscation scheme, the model, the mapping key, and the indices file should be delivered to an end-user trusted neural device (TND) for the correct inference execution.

B. Hardware Support

Modern NPUs, such as Google TPU, ARM Ethos, and Samsung Exynos NPU [10]–[12], implement a systolic array-based design. Weights and inputs are pre-loaded into buffers, and subsequently, with each clock, they are fetched by the processing elements (PEs), multiplied, and accumulated. ARM Ethos [11] carries a weight decode unit responsible for managing the DMA controller's weight stream, decompressing it on-the-fly (reordering, padding, and aligning into blocks), and storing them into double-buffered registers to feed the MAC engine. Samsung Exynos NPU [12] comprises two data-staging units (DSUs) on each NPU core. Weight decompressors units inside DSUs are responsible for decompressing the weight stream and sending it to a proper dispatcher unit. The dispatcher unit selects which weights should be dispatched along with the feature maps into the MAC arrays. Google TPU [10] handle the weights through a weight fetcher unit that reads the stream of data from the DRAM, reorders it, and makes it ready for the MxM unit to consume it.

A custom weight pre-loading step can enable the proposed swap-based model obfuscation technique by leveraging the current weight decoder/decompressor/fetcher unit design. First, the mapping/control key is securely one-time loaded into an NPU register (shared by layers). Then, as layer weights are loaded, the decoder/decompressor/fetcher unit also fetches pairs of indices regarding that layer. Without the correct mapping key, the weight decoder unit will swap the wrong structures and dispatch the wrong weights into the PEs, causing the model to misbehave. The custom placement creates a walled garden ecosystem where only models from verified parties can run on the trusted neural device (TND), closing off the edge-device platform to unauthorized model execution. As specific design details of Ethos's weight decoder, Exynos weight decompressor/dispatcher, and TPU weight fetcher are not publicly available, we only describe this design principle and not the detailed implementation.

IV. Evaluation

To evaluate our proposed model obfuscation technique, we selected three well-known CNN models with different numbers of layers, filters per layer, and filter sizes. The models were pre-trained on the ImageNet ILSVRC-2012 dataset [13] for the image classification task. All the experiments were performed with PyTorch 1.6 framework on a workstation equipped with dual Intel(R) Xeon(R) Gold 6246 CPUs @ 3.30GHz, 256GB RAM and two NVIDIA Quadro RTX 8000 GPUs. To simulate the proposed obfuscation mechanism, we generate a random mapping key that holds how filters, rows, or columns should be swapped. This process is tightly correlated to the model since the random mapping key generator needs prior knowledge about its structure (number of layers, number of filters per layer, and filter size). After loading the pre-trained model, we swap all structures based on the mapping key and store it back into the disk. An adversary in possession of the model's parameters would run inference, but with no evidence that the model has been locked.

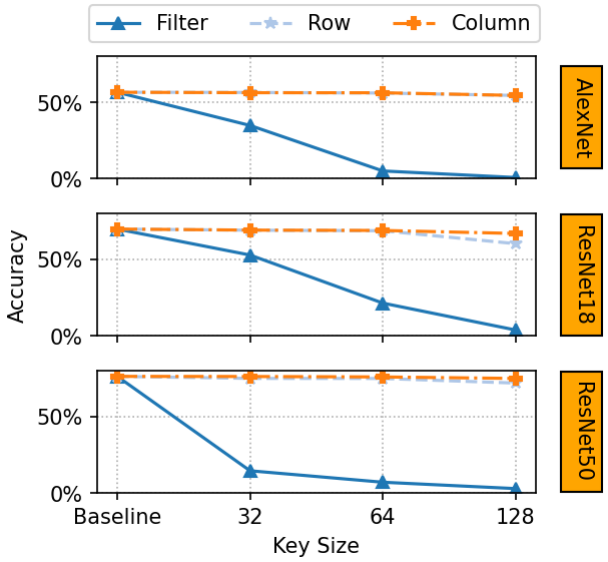

Fig. 3. Effect of the key sizes for AlexNet, ResNet18 and ResNet50 top 1 accuracy under the three proposed model obfuscation techniques.

A. Performance of Obfuscated Model

An obfuscated DCNN should behave in the following way: provide high accuracy, when deployed into a trusted end-user device with the correct key. Any attempt to run the locked model by untrusted parties should be thwarted by dropping the accuracy level by a wide margin. We performed a set of experiments to analyze each structure swap's robustness and impact over the networks. We generate a random key and, based on that information, we swap the filters, rows, or columns within the same layer for each neural network. We compare an unlocked DCNN version (Baseline) over three locked DCNNs with different key sizes: 32, 64, and 128 bits. It is clear in Fig. 3 that filter swaps have a more significant impact on the model's overall accuracy. When considering only 32 bits for filter swaps, the accuracy drops by 21.68%, 16.96%, and 61.64% for AlexNet, ResNet18, and ResNet50. As for row and column swaps, they have a negligible impact ($\leq$ 2%) on the accuracy for up to 64-bit key sizes. When

TABLE I

PERTURBATION RATE ON MODELS OUTPUT DISTRIBUTION USING DIFFERENT TYPES OF OBFUSCATION MODE AND KEY SIZE

Swap Location	Perturbation Rate								
	AlexNet			ResNet18			ResNet50		
	32-bit key	64-bit key	128-bit key	32-bit key	64-bit key	128-bit key	32-bit key	64-bit key	128-bit key
Filter	2.43%	22.28%	31%	1.92%	15.1%	19.3%	52.96%	70.58%	34.97%
Row	0.16%	0.27%	0.43%	0.24%	0.29%	1.67%	0.24%	0.25%	0.53%
Column	0.19%	0.21%	0.44%	0.23%	0.26%	0.49%	0.08%	0.17%	0.28%

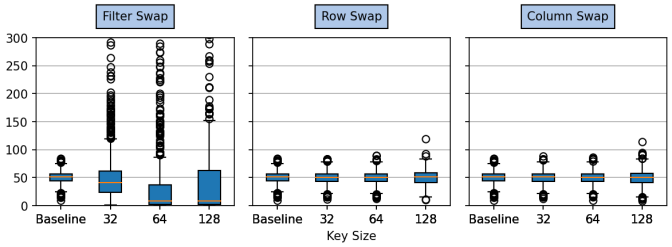

Fig. 4. Impact of model obfuscation techniques over AlexNet output class distribution.

considering a 128-bit key size scenario, rows and columns swap add up to a 10% drop in the overall accuracy.

This behavior occurs because, in a larger group of weights, there is a higher probability of swapping relevant weights with non-relevant ones [14]. Thus, we observe that *filter swap* is the best way to achieve our obfuscation goals.

B. Information Leakage Analysis

If a defender considers accuracy as the only metric for evaluating the obfuscation technique, one could say that the filter swap is the right pick. However, swapping many relevant weights and drastically reducing the accuracy results incurs in *considerable perturbation* in the output class distribution. In this case, by analyzing the outputs of sequential inferences runs, a potential adversary can distinguish between a locked or unlocked model.

To quantify the perturbation rate of each obfuscation model scheme, we propose a new metric (pr) described by Equation 1 which takes into account an optimal distribution ($optimal\_count_i$) for each class in the dataset and a bad distribution (bad_count_i) that arises from a locked model for all available classes in the dataset (N). Therefore, a high pr means the proposed obfuscation scheme incurs a high perturbation of the final distribution. Fig. 4 and TABLE I show how the output class distribution of AlexNet behaves in the previous DCNN performance experiment. The perturbation is apparent when applying filter swap obfuscation, even for the smallest key size ($pr > 22\%$). Row and column swap obfuscation preserve the distribution ($pr < 1.7\%$), with a marginal prediction count increase for few outliers.

$$pr = \frac{\sqrt{\sum_{i=0}^{N}(optimal\_count_i - bad\_count_i)^2}}{total\_count} \quad (1)$$

Considering that filter swap decreases accuracy at an acceptable rate but harms the output distribution and that row/column swaps provide a small degradation on the overall accuracy while keeping the class distribution stable, how can we find the balance between these two strategies? We propose a hybrid obfuscation scheme by merging filter and row/column swaps. Few filters are swapped, and the remaining key bits are reserved for rows or columns swap. Fig. 5 and TABLE II show the impact on accuracy and distribution of different types of hybrid swap configurations. Reserving 20 bits of the key for filter swap is sufficient to drop the accuracy at substantial rates – 17.73%, 31.47%, and 35.38% for Hybrid Row over AlexNet, ResNet18, and ResNet50 with 128-bit key, while keeping **most of** the distribution perturbation at an acceptable rate. As a merit of comparison, the key-dependent backpropagation algorithm [8] illustrated in Fig. 1 incurs a perturbation rate of 18% over a small four-layers deep network (LeNet-4) evaluated with a ten-classes dataset (MNIST).

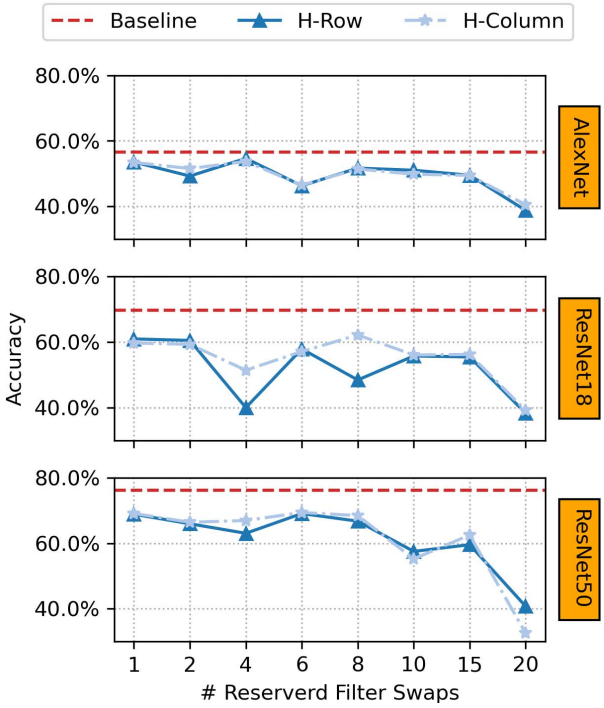

Fig. 5. Impact on models accuracy using different types of configurations for hybrid row (H-Row) and hybrid column (H-Column) modes using 128-bit key.

TABLE III shows the accuracy drop results for both hybrid row and hybrid column approaches with 128-bit key on AlexNet, ResNet18, and ResNet50. By reserving 20 bits of the key for filter swap, we can preserve the output distribution

TABLE II

PERTURBATION RATE ON MODELS OUTPUT DISTRIBUTION USING DIFFERENT TYPES OF CONFIGURATIONS FOR HYBRID ROW AND HYBRID COLUMN MODES USING 128-BIT KEY

# Reserved Filter Swaps	Perturbation Rate 128-bit key					
	AlexNet		ResNet18		ResNet50	
	Hybrid Row	Hybrid Column	Hybrid Row	Hybrid Column	Hybrid Row	Hybrid Column
1	0.56%	0.59%	1.21%	1.47%	0.76%	0.77%
2	0.94%	0.71%	1.26%	1.51%	1.10%	1.01%
4	0.42%	0.51%	10.67%	3.46%	1.47%	1.06%
6	1.30%	1.31%	1.29%	1.34%	0.69%	0.68%
8	0.69%	0.70%	4.24%	0.79%	1.10%	1.90%
10	0.84%	0.90%	1.52%	1.55%	6.78%	7.45%
15	0.87%	0.90%	2.15%	2.27%	2.46%	1.92%
20	**1.95%**	**1.79%**	**6.1%**	**5.58%**	**18.48%**	**26.41%**

TABLE III

SUMMARY OF THE OBFUSCATION RESULTS OVER THE DCNN MODELS THAT COMPOSE THE BENCHMARK SET [1], [15], [16]. ACCURACY IS MEASURED BASED ON TOP-1 ERROR ON IMAGENET [13] DATASET. THE RESULTS CONSIDER AN 128-BIT KEY FOR FILTERS, ROWS, AND COLUMNS SWAPS. THE HYBRID MODE IS COMPOSED OF 20 FILTERS AND 108 ROWS/COLUMNS SWAPS

Model	Original Accuracy	Obfuscated Model 128-bit key									
		Filter		Row		Column		Hybrid Row		Hybrid Column	
		accuracy	% drop	accuracy	% drop	accuracy	% drop	accuracy	% drop	accuracy	% drop
AlexNet	56.52%	0.85%	**55.67%**	54.45%	**2.07%**	54.40%	**2.12%**	38.80%	**17.73%**	40.54%	**15.98%**
ResNet18	69.76%	3.84%	**65.92%**	60.21%	**9.55%**	66.88%	**2.88%**	38.29%	**31.47%**	39.14%	**30.64%**
ResNet50	76.13%	2.85%	**73.28%**	71.75%	**4.38%**	74.77%	**1.36%**	40.75%	**35.38%**	32.56%	**43.57%**

(up to 18.48% considering the Hybrid Row scheme) and reduce the overall accuracy at acceptable rates (more than 35.38% in best cases), providing the best trade-off between accuracy drop and perturbation. Increasing the reserved bit keys for filter swap by more than twenty incurs a much higher distribution perturbation, mainly because the first convolutional layer is more sensitive and less redundant [17], and a simple additional filter swap impacts the model's overall accuracy.

Model providers can make use of the perturbation metric along with the accuracy drop in order to achieve an optimal (or close to optimal) obfuscation scheme. Since the obfuscation process comprises random selections of filters, rows, and columns, an optimization process can deliver even better results than the ones presented in TABLE III.

C. Security Analysis of Swap-Based Technique

A significant challenge for the proposed swap-based technique is to guarantee that even if the potential attacker possesses the model structure, parameters, and list of swap indices, it would not be possible to achieve the same accuracy as trusted parties. To analyze the security properties of the proposed approach, we mount two types of attacks on the locked models in an effort to discover the correct key.

1) Brute-Force Attack: The first type of attack is a brute-force mode with random walks. First, we lock the model with the swap-based obfuscation technique using a 128-bit key. Then, we randomly generate an *initial key*, which results in some accuracy. We then perform a random walk over the individual bits of the initial key, flipping one at a time and checking the new accuracy with a known test set. If the accuracy goes up, we keep the new bit and remove the index from the random search. If the accuracy does not improve, the bit key is reverted, and we perform random walks over the remaining indices. This process is repeated until hitting the accuracy target or after the expiry of some number of iterations (1 000 in this case). A secure technique must not allow the attack to reach the target accuracy by random walk over the key. Further, a secure model should keep the accuracy far from the target by an acceptable margin.

Fig. 6 shows the success potential of the attack on a 128-bit key for each network during 1 000 trials. Each trial corresponds to a random walk step, where one of the key's bit is flipped, and a full prediction over the Imagenet validation set is performed. The three models are capable of sustaining a substantial accuracy drop when applying the filters swap approach. With only 20 bits for filter swaps out of 128, the hybrid row approach maintains the accuracy drop with up to 8.8% for ResNet18.

The models begin at a low accuracy level due to the random generation of the start point key. The accuracy rises after a set of trials, but at some point, it gets stagnant, not exceeding a specific threshold. This phenomenon happens due to the non-uniqueness of the swaps. During the filter and hybrid mode's obfuscation process, the first filter is swapped considering all the available filters/rows/columns in the particular layer. However, the remaining swaps might choose one of the filters

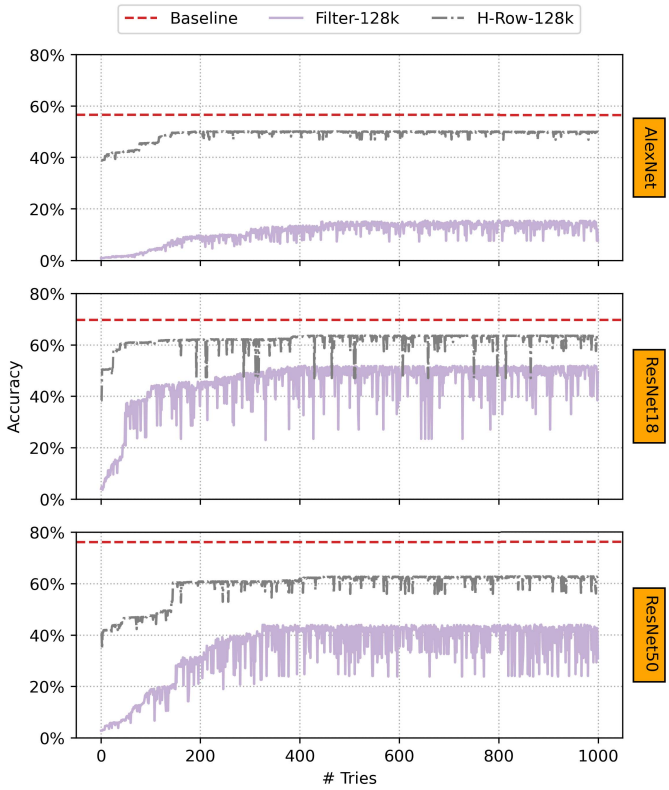

Fig. 6. Security analysis of the swap-based obfuscation technique using filter (Filter-128k) and hybrid row (H-Row-128k) modes with 128-bit key against brute-force attack.

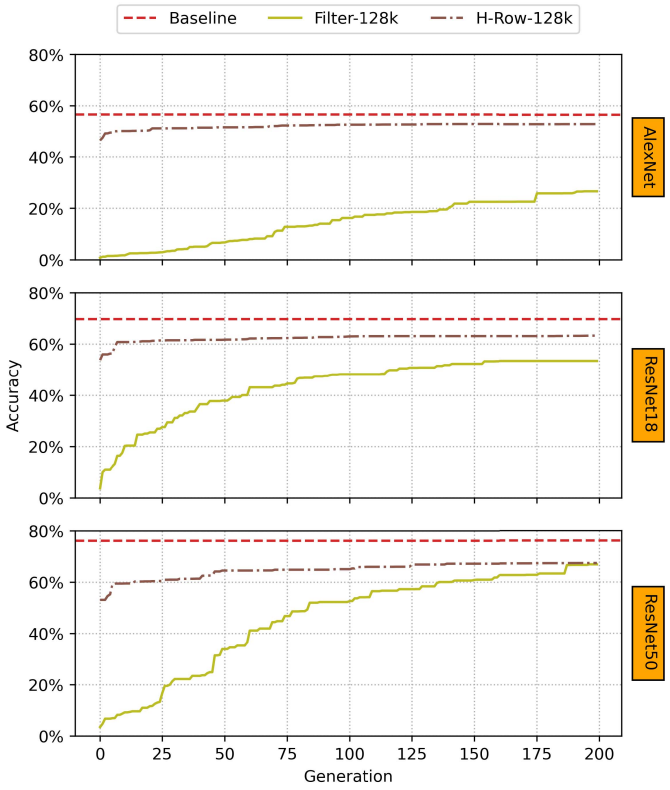

Fig. 7. Security analysis of the swap-based obfuscation technique using filter (Filter-128k) and hybrid row (H-Row-128k) modes with 128-bit key against GA-based attack.

from the previous swaps, creating a relation between the bit keys. With that approach, the Model Provider can determine a specific order in which the bit keys should be flipped to achieve the parameters' correct placement.

2) Genetic Algorithm Attack: The second experiment is a genetic algorithm (GA) based attack, which consists of a genomes population representing possible keys to unlock the model. The genomes are evaluated with a fitness function, consisting of a full prediction over the Imagenet validation set. The resulting accuracy expresses how well a genome performed over the locked model. After generating and evaluating the initial population, an iterative process called generation starts. During each generation, genomes are ranked based on the fitness function. The most relevant ones are randomly selected to combine their genetic information (key bits) through a crossover function. These genome pairs exchange part of their information, generating a new pair of genomes, denominated offsprings. Then, both offsprings suffer mutation where a single or multiple bits can be flipped. Lastly, a new generation starts by evaluating the new genomes population.

The GA-based attack requires a set of parameters to converge appropriately. Population size, number of generations, crossover function, crossover rate, and mutation rate dictates how the GA will behave and is tightly coupled to the problem. Fine-tuning these parameters requires several combinations and runs, demanding a humongous amount of time and computational resources, mainly if the fitness function comprises a thousand DNN inferences. The search for an optimal set of parameters is out of the scope of this work. Due to this fact, we keep the parameters with the following values:

- Genome size: 128
- Population size: 10
- Number of Generations: 200
- Crossover function: Single-point Crossover
- Crossover rate: 0.9
- Mutation rate: 0.5

Fig. 7 shows the success potential of the GA-based attack on a 128-bit key for each network over 200 generations. Each point in the plot corresponds to the top-1 genome accuracy of a generation. Like the brute-force attack, the three models can sustain a substantial accuracy drop when applying the filter swap approach. The hybrid-row mode with 20 bits for filter swaps can maintain a slight margin of 4% accuracy drop for AlexNet after 2 million inference trials.[1]

Note that we decided to run the experiment predictions on the whole validation dataset for simplicity of coding. In a real scenario, a potential attacker would not have this amount of data. Instead, the attacker would have to make several predictions using a small fraction of the dataset ($<$10%). Therefore, this experiment considers an unusual case, where the attacker is assumed to possess the same amount of data as the Model Provider. Thus, we demonstrate that *seeking monotonic accuracy improvement via random walk or using*

[1]# trials = 200 generations $\times$ 10 genomes $\times$ 1,000 images.

TABLE IV
DCNNS TOP-1 ACCURACY UNDER MODEL COMPRESSION. PRUNING (P) REMOVES MORE THAN 59% OF THE PARAMETERS BY ADDING LESS THAN 1% ERROR ON ACCURACY

Model	Model Type	Data Type	Pruning Rate	Top-1 acc.
AlexNet	Original (D)	fp32	-	56.52%
	Pruned (P)	fp32	88.31%	56.61%
ResNet18	Original	fp32	-	69.76%
	Pruned	fp32	59.92%	69.87%
ResNet50	Original	fp32	-	76.13%
	Pruned	fp32	84.57%	75.52%

GA will not uncover the key in the proposed obfuscation approach.

D. Security Enhancements With Pruned Models

A considerable number of the parameters of a DCNNs have little or no impact on the final accuracy of a trained network [18]. Pruning is a compression technique to avoid unnecessary computation and storage by removing these non-relevant weights through an iterative process [19]. Since the swap-based obfuscation heavily relies on the selected weights, pruning should significantly impact the proposed technique's performance.

Fig. 8 show the secure potential of 64-bit and 128-bit keys filters swaps over pruned and dense models (details in TABLE IV) against the brute-force attack performed in Section IV-C.1. Pruned models outperform the original ones (dense), especially with 128-bit key size over ResNet18 and ResNet50. Therefore, pruning can be a potential way to enhance swap-based obfuscation security.

V. SYSTEM OVERVIEW

Distributing a keyed obfuscation model presents a challenge of secure *key* and *model* distribution. This section showcases an end-to-end deep learning trusted system, ranging from secure model distribution to protected model deployment/execution on trusted edge devices. We consider the proposed swap-based obfuscation technique as an additional security layer on top of well-known defense mechanisms that provide secure key distribution.

The overview of an ideal end-to-end deep learning trusted execution system is presented in Fig. 9. The system considers two main parties: *model providers* and *end-users*. Model providers develop, train, and commodify high-accuracy deep learning models. They invest in creating labeled data from multiple sources and in high-performance computing systems for model training. End-users are trusted neural device (TND) holders aiming to run inference applications bought from model providers. The TND has native hardware support for relevant security functions and primitives in the form of a hardware security module (HSM), execution of swap-based obfuscated models within the neural processing unit (NPU), and additional DRAM protection via memory encryption.

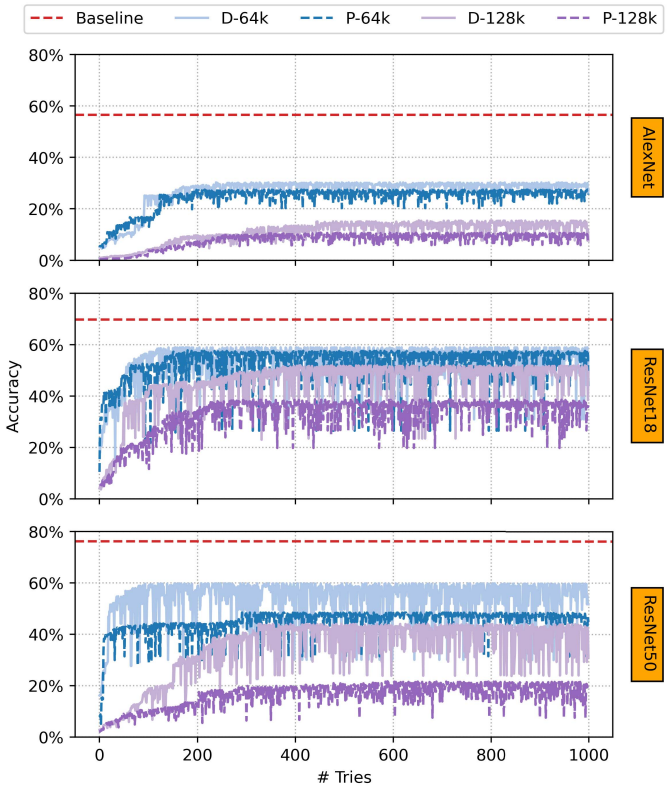

Fig. 8. Security analysis of the swap-based obfuscation scheme over Dense (D) and Pruned (P) models of AlexNet, ResNet18, and ResNet50 using 64-bit and 128-bit keys against brute-force attack.

A. Secure Model Distribution

The first goal for the model provider is to securely distribute their valuable IP to end-users over an untrusted channel to prevent theft while transmission.

The step-by-step protocol for secure model distribution, as illustrated in Fig. 9, is as follows:

1) End-user makes a purchase request for a model from a model provider/vendor and provides their public key, PK_{user}, to the vendor. The public key is generated by the HSM in the TND;
2) The model provider receives the request and the end-user's public key. An obfuscated inference model is generated with a random mapping key, $Key_{obfuscate}$;
3) The model provider/vendor encrypts the obfuscated model with a end-user specific symmetric key, Key_{vendor}. The symmetric key and the obfuscation key, $Key_{obfuscate}$, are both encrypted by the vendor using the end-user's public key, PK_{user}, before transmission;
4) The encrypted model and the encrypted keys are transferred to the end-user TND.

For this work, we assume that elliptic-curve cryptography (ECC) is used to realize the public-key cryptography (PKC). Secure key exchange is performed using the elliptic-curve Diffie–Hellman (ECDH) protocol [20]. We also assume that the HSM generates a 192-bit private key using physically unclonable functions (PUFs). This key is not used for any other purpose besides enabling PKC using ECC. Current best 192-bit ECC hardware implementation in literature requires 83K gates [21]. Since the ECC module is only used for

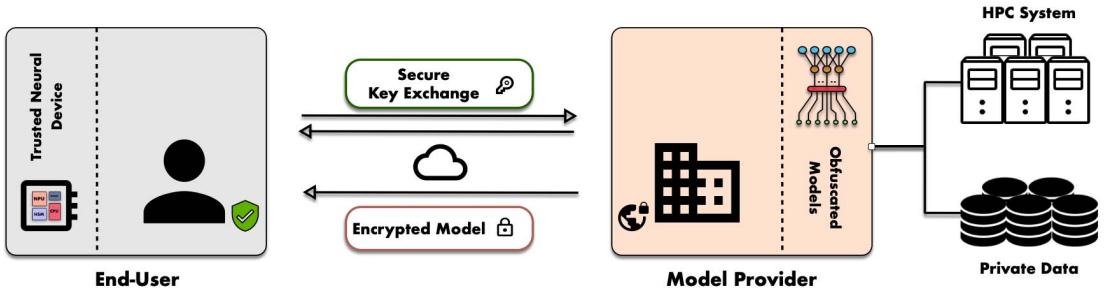

Fig. 9. Overview of secure model distribution.

key exchange during the infrequent secure model distribution, the throughput need not to be high.

B. Memory Encryption

In this section, we will discuss the need for memory encryption of the neural network model during usage in real-time applications and explore the various considerations that need to be taken into account to implement reliable encryption under varying system constraints.

Securing the distribution of machine learning models ensures that valuable IP can be stored safely in non-volatile storage. When needed, the model parameters are loaded on-chip and decrypted to be used by the neural network accelerator. However, for many modern applications, the size of a machine learning model is quite significant and the model cannot be completely contained within the on-chip cache. For example, ResNet50 has over 23 million weight parameters and the total size of the trained model is ≈98 MB [22]. Even if we optimized the weights by lowering the precision to reduce the total space requirement for the model, we still need to account for the storage of input data, temporary values and programming instructions. Furthermore, the TND may run multiple concurrent applications, which increases the space requirements even further. Typically, total on-chip cache sizes are in the 10's of megabytes with even dedicated neural network processing application platforms such as Tesla's self-driving chip having just 32MB SRAM memory [23]. Hence, the model and associated data will need to be stored in an intermediate off-chip memory, like DRAM, and only the required parameters may be accessed as needed.

1) Attack Overview: If the partial model off-loaded to memory is unencrypted, then the attacker can target the memory using a range of physical side-channel, probing or invasive attacks [24]. The introduction of faster non-volatile memory technology such as 3D XPoint to supplement DRAMs further exacerbates the problem as the stored data is persistent. The attacker's goal is to retrieve the weight parameters and architectural information of the neural network model. In this work, our goal for exploring memory encryption is to protect the weight parameters when they are stored in memory. Protection of model architecture is achieved using model obfuscation, as detailed in Section III.

2) System Overview: A memory encryption system is illustrated in Fig. 10. The basic operation is detailed below.

- When a neural network application is invoked, the encrypted model, including the authentication data (digest), provided by the vendor is loaded onto the DRAM memory.
- The encrypted model is loaded onto the TND as needed and decrypted first using the vendor-supplied key, Key_{vendor}, by the HSM. Authentication data or digest supplied with the model is used to verify the model.
- During execution, the model parameters that need to be off-loaded to memory, due to cache overflow, are encrypted with a temporary key, key_i, generated by the HSM. We assume a minimum key length of 128 bits. This key is never exposed outside the TND and is only accessible to the HSM. Multiple such keys may be generated and used during the program execution.
- When the model parameters are loaded back from the memory, the key, key_i, is used to decrypt them.

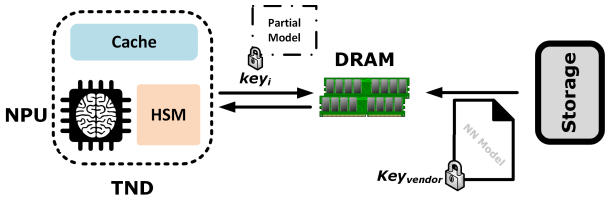

Fig. 10. Illustration of memory encryption system.

If the memory data is lost due to power loss, then the original secure model in the storage can be used to restart the application.

3) Resource Considerations: To implement memory encryption reliably and as ubiquitously as possible, we need to take into account the variability in the available resources across different platforms. Platforms with high resource availability, such as Tesla's FSD [23], can be assumed to have a large cache, high speed memory bus and larger area available to implement HSM features, including dedicated secure storage. Power consumption is also not considered a limiting factor. In contrast, low resource platforms are assumed to have a lower cache size, no secure HSM storage and limited power budget. In both cases, *throughput* is the primary target and will influence the hardware implementation considerations for the encryption cipher. High resource platforms are more likely to run multiple neural network applications and the size of the models can be quite large, requiring frequent memory accesses. For low resource platforms, the limited cache means that only small portions of the model can be loaded at a time and hence, there will be frequent transfer of data between TND and memory.

4) Key Generation: The HSM needs to have minimum capabilities such as support for public-key cryptography (PKC), an encryption cipher implementation and physically unclonable functions (PUFs) to obtain the private key(s). High resource platforms may additionally implement a robust true random number generator (TRNG) for generating the intermediate keys (key_i), while low resource platforms may only depend on the utilization of PUFs to obtain additional keys.

5) Performance Considerations: To enable seamless execution of the inference application while using the memory encryption scheme, it is vital to retrieve the encrypted model parameters from memory and decrypt them for use to prevent stalling the application. Model parameter encryption can be done more slowly as it has no effect on inference. However, the primary bottleneck for memory encryption is the $>$80Gbps read/write bandwidth obtained by modern DDR4 and LPDDR4 memories [25]. The GCM mode of operation of AES cipher has been adopted for its performance over other modes. The most efficient implementation can achieve $\approx$150Gbps, but has a large area cost [26]. Hence, it is only suitable for high resource platforms. For low resource platforms, lightweight block cipher such as PRINCE are more suitable due to its $14-16\times$ smaller size compared to AES [27]. It is possible to use the fact that model execution sequence is usually fixed to ensure that the parameters that are required from the DRAM memory in the near-future are pre-loaded and decrypted to ensure they are ready for use.

VI. Conclusion

In this paper, we presented a deep-learning IP protection solution for NPU edge devices. Our solutions encompass a novel, lightweight, scalable model obfuscation technique that *does not require a specialized training process*. The proposed obfuscation technique can hide the original network's internal structure by swapping rows, columns, and full filters based on a secret mapping key. We demonstrate the effectiveness of our approach across different DNN architectures. Moreover, we verify that our hybrid scheme avoids information leakage by preserving the output class distribution. The swap-based obfuscation technique is the only solution known to date that is scalable and does not require retraining for secret key changes. Further, the solution allows model updates by the model provider without requiring any additional features.

Lastly, we described an end-to-end deep learning trusted execution system, ranging from secure model distribution via hardware root-of-trust and public-key cryptography infrastructure (PKI) to protected model deployment/execution on trusted edge devices via low-latency memory encryption solution for real-time DNN execution. This showcases working of the entire system.

References

[1] 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.

[2] Y. Xu, A. Sep, Y. Ban, R. Horaud, L. Leal-Taixe, and X. Alameda-Pineda, "How to train your deep multi-object tracker," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 6787–6796.

[3] T. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. Red Hook, NY, USA: Curran Associates, 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[4] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Conf. Secur. Symp.*, 2016, pp. 601–618.

[5] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 36–52, doi: 10.1109/SP.2018.00038.

[6] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," 2020, *arXiv:2009.00993*. [Online]. Available: http://arxiv.org/abs/2009.00993

[7] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *Proc. 55th Annu. Design Autom. Conf.* New York, NY, USA: Association Computing Machinery, Jun. 2018, doi: 10.1145/3195970.3196105.

[8] A. Chakraborty, A. Mondal, and A. Srivastava, "Hardware-assisted intellectual property protection of deep learning models," in *Proc. 57th ACM/EDAC/IEEE Design Automat. Conf. (DAC)*. Piscataway, NJ, USA: IEEE Press, 2020.

[9] H. Xu, Y. Su, Z. Zhao, Y. Zhou, M. R. Lyu, and I. King, "DeepObfuscation: Securing the structure of convolutional neural networks via knowledge distillation," *CoRR*, vol. abs/1806.10313, 2018. [Online]. Available: http://arxiv.org/abs/1806.10313

[10] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.* New York, NY, USA: Association Computing Machinery, 2017, p. 1–12, doi: 10.1145/3079856.3080246.

[11] A. Skillman and T. Edso, "A technical overview of the arm Cortex-M55 and Ethos-U55: ARM's most capable processors for endpoint AI," in *Proc. Hot Chips 32, Symp. High Perform. Chips*, 2020, pp. 1–20.

[12] J. Song *et al.*, "An 11.5TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile SoC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 130–132.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. CVPR*, Jun. 2009, pp. 248–255.

[14] Y. L. Cun, J. S. Denker, and S. A. Solla, *Optimal Brain Damage*. San Francisco, CA, USA: Morgan Kaufmann, 1990.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds. San Diego, CA, USA: CoRR, 2015, pp. 1–14. [Online]. Available: http://arxiv.org/abs/1409.1556

[17] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, vol. 1. Cambridge, MA, USA: MIT Press, 2015, pp. 1135–1143.

[18] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*. San Francisco, CA, USA: Morgan Kaufmann, 1990, pp. 598–605.

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. ICLR Int. Conf. Learn. Represent.*, 2016, pp. 1–13.

[20] *Elliptic-Curve Diffie–Hellman (ECDH) Protocol*. Accessed: Dec. 14, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Elliptic-curve_Diffie-Hellman

[21] L.-Y. Yeh, P.-J. Chen, C.-C. Pai, and T.-T. Liu, "An energy-efficient dual-field elliptic curve cryptography processor for Internet of Things applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 9, pp. 1614–1618, Sep. 2020.

[22] *Keras Applications—Model Sizes*. Accessed: Dec. 14, 2020. [Online]. Available: https://keras.io/api/applications/

[23] E. Talpes *et al.*, "Compute solution for Tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, Mar. 2020.
[24] M. Isakov, L. Bu, H. Cheng, and M. A. Kinsy, "Preventing neural network model exfiltration in machine learning hardware accelerators," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2018, pp. 62–67.
[25] *DDR4 SDRAM*. Accessed: Dec. 14, 2020. [Online]. Available: https://en.wikipedia.org/wiki/DDR4_SDRAM
[26] S. Ghosh, L. S. Kida, S. J. Desai, and R. Lal, "A >100 Gbps inline AES-GCM hardware engine and protected DMA transfers between SGX enclave and FPGA accelerator device," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 178, Feb. 2020.
[27] J. Borghoff *et al.*, "Prince—A low-latency block cipher for pervasive computing applications," in *Advances in Cryptology—ASIACRYPT 2012*. Berlin, Germany: Springer, 2012, pp. 208–225.

Brunno F. Goldstein (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the COPPE-System Engineering and Computer Science Program, Federal University of Rio de Janeiro, Brazil, under the supervision of Prof. Felipe M. G. França and Prof. Sandip Kundu. His current research interests include hardware reliability, machine learning, and hardware security.

Vinay C. Patil (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Massachusetts Amherst, Amherst, MA, USA, under the supervision of Prof. Sandip Kundu. His research interests include VLSI circuit design, hardware security, machine learning, and neuromorphic computing.

Victor C. Ferreira is currently pursuing the Ph.D. degree with the COPPE-System Engineering and Computer Science Program, Federal University of Rio de Janeiro, Brazil, under the supervision of Prof. Felipe M. G. França and Prof. Alexandre S. Nery. His current research interests include computer architecture, hardware reliability, and machine learning.

Alexandre S. Nery (Member, IEEE) received the M.Sc. and D.Sc. degrees in systems engineering from the Federal University of Rio de Janeiro (UFRJ), Brazil, in collaboration with the Eindhoven University of Technology (TU/e), The Netherlands, in 2010 and 2014, respectively. He worked as a Visiting Researcher with Intel UMG (former Silicon Hive) on hardware sharing and automatic custom instruction identification using silicon hive reconfigurable VLIW architecture with the UFRJ, in collaboration with TU/e. From 2015 to 2018, he worked as an Associate Professor of computer science with the Rio de Janeiro State University (UERJ). Since 2018, he has been an Associate Professor of network engineering with the University of Brasília (UnB). His research interests include reconfigurable computing (FPGA accelerators), distributed computing (cloud/edge/fog/in-situ architectures), and more recently, cybersecurity.

Felipe M. G. França (Senior Member, IEEE) received the B.Sc. degree in electrical engineering and the M.Sc. degree in computer science from the Federal University of Rio de Janeiro in 1982 and 1987, respectively, and the Ph.D. degree in neural systems engineering from the Imperial College of Science Technology and Medicine in 1994. He is currently a Visiting Professor with the Federal University of Rio de Janeiro. He has experience in computer science and electronics engineering, acting on the following subjects, namely artificial neural networks, complex systems, computer architecture, distributed algorithms, computational intelligence, collective robotics, complex systems, and intelligent transportation systems.

Sandip Kundu (Fellow, IEEE) was the Program Director of the Division of Computer and Network Systems, National Science Foundation, a Principal Engineer at Intel Corporation, and a Research Staff Member at IBM Corporation. He is currently a Professor of electrical and computer engineering with the University of Massachusetts Amherst, Amherst. He has published more than 200 articles in VLSI design and CAD, holds 12 patents, and coauthored multiple books. He served as the Technical Program Chair for ICCD in 2000, the Co-Program Chair for ATS in 2011, ISVLSI in 2012 and 2014, and DFT in 2014. He has been a Distinguished Visitor of the IEEE Computer Society.