IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 1, pp. 506–556. DOI:10.46586/tches.v2022.i1.506-556

ModuloNET: Neural Networks Meet Modular Arithmetic for Efficient Hardware Masking

Anuj Dubey[1], Afzal Ahmad[2], Muhammad Adeel Pasha[3], Rosario Cammarota[4] and Aydin Aysu[1]

[1] North Carolina State University, Raleigh, US, {aanujdu,aaysu}@ncsu.edu
[2] The Hong Kong University of Science and Technology, Hong Kong, afzal.ahmad@connect.ust.hk
[3] Lahore University of Management Sciences, Lahore, Pakistan, adeel.pasha@lums.edu.pk
[4] Intel Labs, San Diego, US rosario.cammarota@intel.com

Abstract. Intellectual Property (IP) thefts of trained machine learning (ML) models through side-channel attacks on inference engines are becoming a major threat. Indeed, several recent works have shown reverse engineering of the model internals using such attacks, but the research on building defenses is largely unexplored. There is a critical need to efficiently and securely transform those defenses from cryptography such as masking to ML frameworks. Existing works, however, revealed that a straightforward adaptation of such defenses either provides partial security or leads to high area overheads. To address those limitations, this work proposes a fundamentally new direction to construct neural networks that are inherently more compatible with masking. The key idea is to use modular arithmetic in neural networks and then efficiently realize masking, in either Boolean or arithmetic fashion, depending on the type of neural network layers. We demonstrate our approach on the edge-computing friendly binarized neural networks (BNN) and show how to modify the training and inference of such a network to work with modular arithmetic without sacrificing accuracy. We then design novel masking gadgets using Domain-Oriented Masking (DOM) to efficiently mask the unique operations of ML such as the activation function and the output layer classification, and we prove their security in the glitch-extended probing model. Finally, we implement fully masked neural networks on an FPGA, quantify that they can achieve a similar latency while reducing the FF and LUT costs over the state-of-the-art protected implementations by 34.2% and 42.6%, respectively, and demonstrate their first-order side-channel security with up to 1M traces.

Keywords: Neural networks · Side-channel attacks · Hardware Masking

1 Introduction

Illicit extraction of proprietary machine learning (ML) models has become a serious concern in the ever-growing ML industry. Model owners can host a trained ML model either on a cloud server or on an edge device and provide a publicly accessible interface for predictions. Since designing and training an ML model is an expensive process, the trained model has a monetary value associated with it. It has been reported that a single trained ML model can cost as much as \$10M [Mag20]. Therefore, the model owner typically charges the users to use the model or purchase the device with the trained model [Sec20]. In both cases, the ML model carries a business value and should be kept confidential from the users.

Recent advances in ML algorithms and the semiconductor industry have enabled efficient deployment of ML models with reasonably high accuracy on resource-constrained edge devices. Edge-based inference obviates constant communication with the cloud,

Received: 2021-07-15 Accepted: 2021-09-15 Published: 2021-11-19

thus improving the inference latency and ensuring data privacy [ZCL+19]. Indeed, the number of edge-based artificial intelligence (AI) chips is expected to be doubled by 2024 [Ins20]. Despite these advantages, it becomes harder to ensure the confidentiality of the deployed model on the edge because the device operates in an environment where physical side-channels attacks like power/electromagnetic (EM) become more applicable.

Physical side-channel attacks are effective and hard to mitigate, primarily because they are based on the device's intrinsic properties, like the data-dependent CMOS power consumption [KJJ99]. Such attacks have been extensively studied over the last two decades in the context of cryptographic implementations [KJJ99, CEvMS15, KGB+18]. Correspondingly, the research on mitigating such attacks has also matured significantly, starting from the empirically secure primitives [AG01, TKL04] earlier to the provably secure primitives at present [ISW03, NRR06, GMK16, GM18, GIB18]. We envision a similar trend for the side-channel research on ML accelerators. Several recent works already demonstrate successful power/EM side-channel attacks on software and hardware implementations of an ML model [BBJP19, YKO+20, WLL+18, YMY+20, MBC21, JYI+20, TSSL20]. There have also been a few proposals on building side-channel defenses against such attacks using the well-known hiding and masking techniques [DCA20b, DCA20a].

The differences between ML and cryptographic algorithms cause challenges when adapting side-channel defenses towards ML. A major problem is due to the integer arithmetic used in neural network computations vs. modular arithmetic in cryptography. Integer arithmetic impedes the direct application of arithmetic masking in the neural network computations due to the leakage of the sign bit [DCA20b]. Prior work investigated this problem and proposed hiding the sign bit leakage through differential circuit styles [DCA20b]. Another work addressed it by decomposing the computations at the bit-level and applying Boolean masking on each operation [DCA20a]. While hiding requires precise control of the back-end flow (and still has some disputed claims [ISU18]), gate-level Boolean masking does not have this drawback but incurs significant overheads.

Motivation and Novelty. Both prior works approach the problem by changing defenses to fit the ML algorithm. By contrast, we explore the alternative—changing the ML frameworks to ease the defense's implementation. Specifically, we reformulate the neural network inference with modular arithmetic (rather than integer arithmetic) and design the building blocks for ML to work with modular arithmetic while preserving the model's accuracy. We call this neural network ModuloNET. We then apply efficient masking styles for different parts of ModuloNET and develop masked components unique for ML topologies. The results show that our solution combines the best of both worlds: it is significantly more secure against first-order attacks than the defense with hiding [DCA20b] and considerably less costly than the only-Boolean masked solution [DCA20a].

Contributions. A summary of our contributions and key results are as follows.

- We address the fundamental incompatibility of neural networks to cryptographic defenses by proposing an alternative way to construct inference that works on modular arithmetic with negligible accuracy loss: less than 0.5% for binarized multi-layer perceptron (MLP) on MNIST, and less than 1% for convolutional neural network (ConvNet) on CIFAR-10 and CIFAR-100 datasets. We identify the challenges of using modular arithmetic in activation, batch normalization (BN) and output layers, and propose a novel layer architecture to address them.
- We implement a parameterized fully-masked inference hardware in which the number of nodes within each layer can be tuned. We opportunistically apply suitable masking styles whenever they become efficient—arithmetic masking for fully connected layers and Boolean masking for activation layers, comparisons, and multiplexers. We

implement a novel and efficient masked thresholder design to securely evaluate the activation function. We also design a novel masked comparator to securely perform masked output layer computations. We implement secure Boolean-to-arithmetic and arithmetic-to-Boolean conversion circuits to switch between different masking styles. We also implement a binarized ConvNet design and show how to mask the (linear) convolution and (non-linear) maxpool operations. This demonstrates the scalability of our proposed techniques to more advanced architectures. The implementation results for the MLP design show that the proposed solution occupies 34.2% and 42.6% fewer LUTs and FFs while achieving virtually the same latency[1] compared to the Boolean-only masked solution [DCA20a].

- We perform leakage evaluation tests of our neural network hardware implementation using the widely accepted TVLA methodology [GGJR11] on an FPGA with power measurements. We first evaluate the security of each masked primitive as a standalone unit and then evaluate the fully masked implementation using those primitives. We demonstrate a first-order security with 1M traces each in fixed and random datasets, both for the individual masked components as well as the fully composed design.

- We define the side-channel security for the ML-specific primitives using the glitch-extended probing model [RBN$^+$15] in which the power and EM side-channel attacks become equivalent. We prove the security of the proposed masked gadgets for ML-specific operations like the activation function and output layer in this model.

Organization. Section 2 outlines our threat model. Section 3 presents relevant background information regarding our notation, hardware masking, and neural networks. Section 4 describes the proposed layer architecture and the software implementation of ModuloNET. Sections 5 and 6 discuss the hardware design of the baseline MLP and masked MLP. Section 7 discusses the design of the baseline and masked ConvNet designs. Section 8 presents the security proofs for the masked gadgets. Section 9 presents our proposed ML model accuracy, hardware implementation results, and leakage evaluation results. Section 10 discusses extensions and future applications of this work and Section 11 concludes the paper.

2 Threat Model

We follow the threat model adopted in prior side-channel works on ML model stealing [DCA20b, YKO$^+$20]. Here, we summarize and highlight the key aspects. The neural network model is trained offline, and the trained model parameters are securely stored in the accelerator before deployment. The device is then deployed in an untrusted environment where it performs inferences. The attacker can obtain power/EM traces from the device while running inference, either via direct access or remotely [ZS18, SGMT18, WLL$^+$18]. We assume a chosen-plaintext attack model where the attacker sends the inputs to the device for classification and can capture multiple traces corresponding to different inputs. The adversary then executes a power/EM-based side-channel attack with the captured traces [KJJ99, BCO04, CRR03]. Furthermore, we present our security proofs in Section 8 using glitch-extended probes first introduced by Reparaz *et al.* [RBN$^+$15].

Following earlier works on model extraction [JSMA19, DCA20b, JCB$^+$20], we assume that the hyperparameters of the model like the types/number of neural network layers are either public or obtained by another attack [PMG$^+$17, YMY$^+$20]. Thus, the adversary tries to extract the model *parameters*, not hyperparameters in our threat model. Also, the number of parameters is in billions while the number of hyperparameters is in hundreds

[1]Refer to Table 2.

making parameters much more critical, hence lucrative in an ML IP. In fact, even if the adversary has access to the hyperparameters, it might still not be able to train the model without investing in the training data set, which is usually not free, or in the compute resources such as a GPU farm. Our work provides confidentiality for the *parameters*. Hence, the proposed defense's goal is to ensure that no information about parameters is leaked during any intermediate computation through a first-order power/EM-based side-channel attack. Therefore, the proposed masking scheme masks *all* intermediate computations and not just, say, the first and last layer. Although this may be another viable option (as in masking the first and last round of AES [SP06]) to reduce masking overheads, recent works have shown that deeper computations can be targeted with ever-evolving attacks [BBH+19].

We analyze binarized neural networks (BNNs) [CB16]. Following prior works on protecting ML hardware [DCA20b, DCA20a], our hardware implementation is constant-time/flow, making it immune to other side-channel attacks such as timing or the access-pattern based [HZS18]. Our design stores all the parameters of the model in the on-chip FPGA block RAMs. Therefore, the hardware implementation is also immune to any memory-based side-channel attacks. We exclude invasive attacks like voltage/EM fault-injection attacks in our threat model [BJH+21, TG20]. We also exclude template/profiled attacks for this work.

3 Background

In this section, we establish the notation that we utilize throughout the paper for ease of understanding. We also introduce the basics of neural networks, BNNs, and hardware masking. These fundamentals would serve as the groundwork for the reader in understanding our proposed methods in future sections.

3.1 Notation

We denote vectors $\mathbf{x}$ with lowercase bold, individual elements of the vectors with italicized subscripts x_i, matrices $\mathbf{X}$ with uppercase, bold roman letters, and matrix elements $x_{i,j}$ at position i, j, where i and j correspond to the row and column also with italicized subscripts. Every scalar x is non-binary unless explicitly stated as $x \in GF(2)$. We use bracketed subscript $x_{[i]}$ to index the i^{th} bit of a scalar x, bar on the top $\overline{x}$ to represent a bit-wise inverse, superscript x^i to refer to the i^{th} share of a masked variable x, calligraphic fonts $\mathcal{O}$ to denote sets, and typewriter font `F(.)` to denote functions. We use braces in superscript $w_{i,j}^{\{k\}}$ to index the variables in different layers of the neural network.

We denote a multiplexer function in hardware design sections as $\texttt{MUX}(.)$, which is defined as follows.

$$\texttt{MUX}(a, b; s) = \begin{cases} a, & s = 0 \\ b, & s = 1 \end{cases}$$

Modulo operation $x \bmod K$ is defined as $x \bmod K \triangleq x - K \left\lfloor \frac{x}{K} \right\rfloor \in [0, K)$. We always use the letter K to denote the modulus. We define $\texttt{DOM}(.)$ (a type of masked AND gate) as $\texttt{DOM}(a^1, a^0, b^1, b^0, r) = (c^1, c^0)$ where $\bigoplus_i c^i = a \cdot b$, $\bigoplus_i a^i = a$, and $\bigoplus_i b^i = b$. $a^i, b^i, r \in GF(2) \forall i$ and r is a fresh and uniformly sampled bit. The $\oplus$ and $\cdot$ symbols represent the bitwise exclusive-OR and AND operations, respectively.

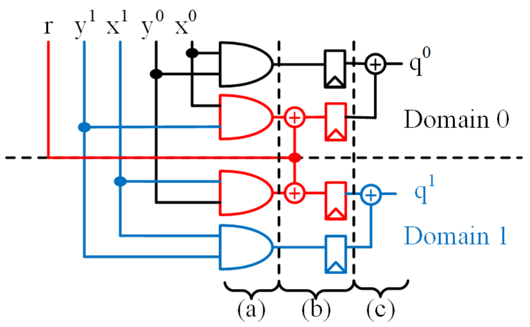


Figure 1: The design of a first-order pipelined DOM-indep multiplier [GMK16]. The components in black and blue belong to domains 0 and 1 respectively. The red colored components represent the cross-domain terms. (a), (b), and (c) refer to the *calculation*, *resharing*, and *compression* steps, respectively.

3.2 Hardware Masking for Side-Channel Security

Basics of Masking. Masking is a side-channel countermeasure that splits the secret variable into multiple, statistically independent shares and transforms the original function to work on these shares independently. In a d-th order masking scheme, the secret x is commonly split into $d+1$ shares $x^0, x^1, \cdots x^d$ such that $x = \bigoplus_i x^i, 0 \leq i < d+1$, and $\bigoplus$ is an addition in the field. The d shares $x^0 \cdots x^{d-1}$ can be sampled uniformly and the $(d+1)^{th}$ share can be created as $x \oplus (\bigoplus_i x^i)$, $0 \leq i < d$. This type of masking is typically called arithmetic masking for non-binary numbers. In GF(2), the additions become exclusive-OR and masking is commonly called Boolean masking. It is efficient to use arithmetic masking to mask arithmetic operations and Boolean masking to mask Boolean operations. Prior work on multiplicative masking of AES demonstrates the efficiency of such implementations [DRB18].

The most critical requirement for a masking scheme is to always keep the shares separated during computations. This is hard to achieve in practice due to glitches that can temporarily recombine the shares and leak the secret [MPO05]. Glitches can be reduced by inserting registers on data paths involving multiple shares [AGM+09]. Threshold Implementation (TI) is a theoretical solution to address the issue of glitches by enforcing the (*non-completeness*) property that none of the intermediate computations (a.k.a., component functions) involve manipulating all the shares of a secret simultaneously [NRR06]. A d^{th} order TI typically requires $td+1$ shares, where t and d are the algebraic degree of the function and the protection order, respectively. Some follow-up works tried to provide security in the presence of glitches using only $d+1$ shares [DRB+16, GMK16].

Domain Oriented Masking. Figure 1 shows a first-order secure DOM-indep multiplier, which assumes the inputs to be independent [GMK16]. The key idea behind DOM is to ensure that in a d^{th} order masked circuit that is split into $d+1$ independent circuits, each circuit only processes at most one share per secret variable. Each share index is associated with a so-called *domain*. A circuit that only receives one share per secret variable cannot leak the secret because every secret share is independent of the secret variable. A cross-domain term introduced due to a non-linear function can potentially leak the secret. Linear functions are affine, thus easy to mask without additional randomness. Figure 1 shows the three steps of a DOM-indep computation. The *calculation* step (a) computes all the partial products, the *resharing* step (b) integrates the cross-domain terms into the primary domains by remasking with a random bit r, and finally, the *compression* step (c) combines the terms into two output shares to prevent the otherwise unnecessary increase in the number of domains. A register at the end of *resharing* step prevents any glitches to propagate to the *compression* step. We choose DOM instead of TI for this work because it provides security in presence of glitches using only uses d+1 shares and because of its widespread adoption in recent masking literature [FBR+21, ABP+18]. We discuss some more optimized alternatives in Section 10.

Theoretical Attack Models. The early research on hardware masking was missing a theoretical framework to model a side-channel adversary capabilities and the properties of the target platform. Thus, it was difficult to provide concrete security guarantees for a newly proposed masking scheme. Ishai *et al.* provided for the first time a theoretical framework to evaluate side-channel countermeasures, which is better known as the *t-probing model* [ISW03]. In this model, the adversary can observe the values of at most t wires in the masked circuit; the circuit is said to be secure if and only if the value on each of the t wires can be simulated using *only* randomness.

The *t-probing* model is a good starting point but it does not model the physical faults in hardware like glitches, transitions and coupling, which can potentially lead to side-channel leakages [MPO05]. Hence, the *t-probing* model was extended to the *robust-probing* model by Faust *et al.* [FGP+18], which also models physical faults in hardware. One of the key ideas in the robust-probing model was to use the so-called *glitch-extended* probes first introduced by Reparaz *et al.* [RBN+15]. These probes leak the value of not only the probed wire but all the wires in the fan-in until the last synchronisation point.

3.3 Neural Networks

Neural networks are a class of ML algorithms used for predictive modeling of data that have recently gained popularity due to their ease of use and effectiveness. They model an unknown function by analyzing a set of known data points (called the training dataset) and make predictions on new data (called the test dataset) using the modeled function.

A neural network consists of multiple neurons organized in layers working in a feed-forward fashion where one layer takes inputs from the previous layer, applies a series of operations, and feeds the outputs to the next layer. The fundamental unit of a neural network is called neuron. A network in which each neuron in a layer is connected to all the neurons in the next layer is often called a multi-layer perceptron (MLP). The layers in an MLP are intuitively termed as *fully-connected* (or *dense*) layers. Neurons multiply their inputs with layer weights and pass the sum to an activation function. Each incoming connection of a neuron has a weight value associated with it. The neuron performs a weighted summation of the inputs and computes the probability using a non-linear function like sigmoid, rectifier linear unit (ReLU), etc. Operations of all neurons within each layer can be condensed using matrix representation as

$$\mathbf{y} = \sigma(\mathbf{x} \cdot \mathbf{W} + \mathbf{b}),$$

where $\mathbf{x}$, $\mathbf{W}$, and $\mathbf{b}$ are input feature, layer weight and bias tensors, respectively. Feature vector, $\mathbf{x} = \{x_m\}$, for $m \in \mathbb{N} \cap [1, M]$, denotes the representations input to a layer, where M is the size of feature vector. Weight matrix, $\mathbf{W} = \{w_{m,n}\}$, and bias vector, $\mathbf{b} = \{b_n\}$, for $n \in \mathbb{N} \cap [1, N]$, represent the learned parameters of the layer, where N is the number of nodes in the layer. The connection weight matrix, $\mathbf{W}$, is calibrated during the training phase by feeding inputs and their corresponding labels, also known as ground truth data, to the network and using a method called backpropagation to iteratively revise the weights such that the network starts predicting the correct labels [Lin76]. Thus, the weights are *crucial* in deciding the model accuracy and hence the most lucrative target in an ML model extraction, which should be protected against side-channel attacks. Similar to weights, other parameters, like bias vectors, $\mathbf{b}$, and Batch-normalization (BN) [IS15] parameters are also computed during training and stored offline for inference. BN is applied to the output of neurons of the layers of the neural network as a form of normalization of layer outputs. $\boldsymbol{\mu}, \boldsymbol{\sigma}$ are the mean and standard deviation of the neuron outputs and $\boldsymbol{\gamma}$, $\boldsymbol{\beta}$ are learnable parameters and are used to shift the mean and standard deviation, respectively. BN parameters are less critical compared to weights of a neural network but they may also be kept confidential.

Given the fact that the layers of neural networks are stacked together with the outputs of one layer being used as the inputs of the next layer, the distribution input to each layer may change with each successive layer, causing the learning algorithm to continuously chase a moving target. Batch-normalization (BN) is a form of standardization used to homogenize the distribution fed to each layer. BN significantly reduces the training time and stabilizes the training by normalizing inputs to each layer [IS15]. Given its wide ranging benefits from allowing training deeper networks to accelerated training, BN is one of the core fundamental operations utilized in neural networks.

Convolutional Neural Networks Fully-connected (FC) layers have several limitations that hamper the scalability and computational efficiency. First, the number of parameters keep increasing with the number of input features, which is a problem if the dataset has a lot of features (e.g., high-resolution images). Second, connecting all the previous layer neurons to the next layers neurons may lead to overfitting of data because every pixel acts as an input feature to every neuron of the next layer. Furthermore, fully-connected layers do not exploit spatial correlation between pixels of input images since each pixel is connected to every other pixel in the image with equal importance. A convolutional (conv) layer addresses these problems by processing smaller regions of the image called the *receptive fields* to compute the output features [FMI83, LHBB99]. It preserves the spatial relations between different parts of an input image and summarizes the features in a concise manner owing to the small receptive fields. It exploits the fact that each pixel and its neighborhood are semantically linked and meaningful. Furthermore, it also promotes translational invariance: the same object can appear anywhere within the image. The number of parameters in a conv layer does not increase with increasing input features, and unlike FC layers, only a subset of input features contribute to an output feature.

Every conv layer has a set of tensors called *kernels* that it uses to perform an inner product with the shifting receptive fields in the input image. The inner product operation is commonly called *convolution*. If we denote the input and output tensors of the convolution layer as $\mathbf{X}$ and $\mathbf{Y}$, respectively, then the element at position (p, q, r) in $\mathbf{Y}$ is computed as

$$Y_{p,q,r} = \sum_{i=1}^{n_h} \sum_{j=1}^{n_v} \sum_{k=1}^{n_i} X_{p+i,q+j,k} \times K_{i,j,k,r},$$

where n_h, n_v, and n_i are the width, height, and depth of the kernel respectively. ConvNets are commonly seen in computer-vision applications. The kernels are learned through training and thus, should remain confidential.

Conv layers are usually followed by a pooling layer that aims to downsample the output retaining only important features. This helps to regularize the data and prevent positional dependence of the detection, i.e., the network detecting a certain feature only if it is present at a specific location. Two common ways to pool the data are either selecting the maximum (a.k.a, *maxpool*), or computing the average (a.k.a. *avgpool*). We choose to implement maxpool for our work because it is hardware-friendly for BNNs [UFG+17].

ConvNets gained popularity after AlexNet [KSH17], a deep ConvNet that demonstrated remarkable results on the ImageNet dataset, the largest computer-vision benchmark, at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [RDS+15]. Since then, ConvNets are considered the de-facto set of algorithms for visual perception tasks and are extensively utilized in a huge number of applications including self-driving vehicles, autonomous robots, and unmanned aerial vehicles (UAVs).

Binarized Neural Networks. The weights and activations in a typical neural network are 32/64-bit floating-point (FP) numbers with high storage and computational needs. Thus, such networks are not suitable for resource-constrained edge devices with low-power and storage requirements. To reduce the memory footprint and computation costs, quantized neural networks with low-precision parameters have been proposed [HCS+17].

BNNs [CB16] use binary weights and activations, and are a natural progression in extreme quantization to reduce the compute and memory footprints of deep neural networks (DNNs). They utilize bit-wise arithmetic operations such as XNOR-POPCOUNT (`XP`), which are extremely efficient compared to matrix multiply-accumulate operations in high-precision DNNs. `XP` on a BNN layer $\boldsymbol{l} = [a_0, a_1, \cdots a_{N-1}]$ with binary features a_i is defined as

$$\texttt{XP}(\boldsymbol{l}) = 2 \times \sum_{i=0}^{N-1} (w_i \overline{\oplus} a_i) - N,$$

where $\times$ and $\overline{\oplus}$ are multiplication and XNOR operations, respectively.

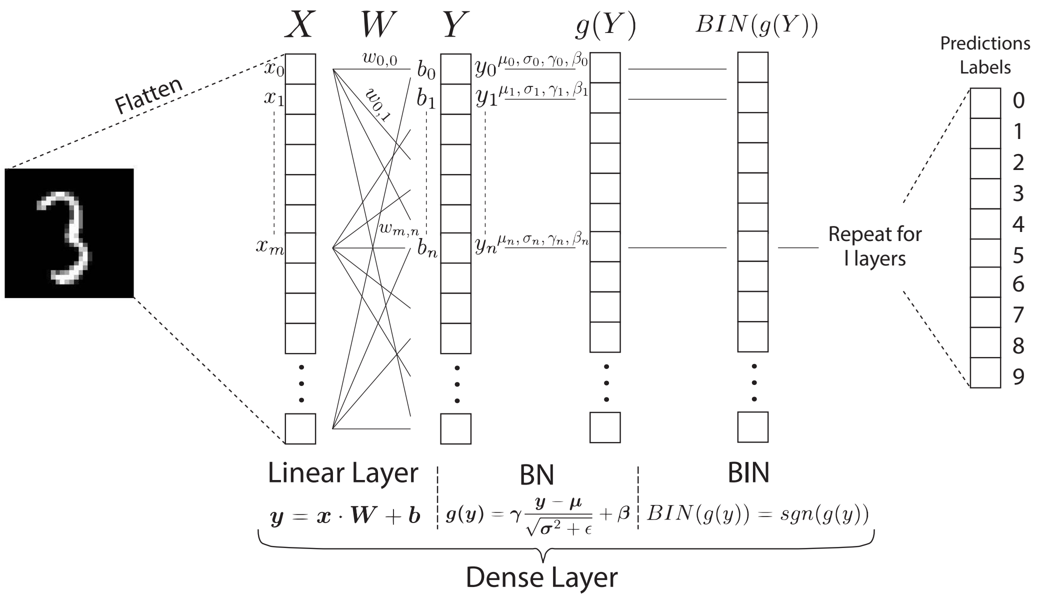
Prior works have demonstrated the computational efficiency of XNOR-POPCOUNT based arithmetic operations in DNNs. For instance, a GPU kernel exploiting the XNOR-POPCOUNT operations can achieve 23× faster matrix multiplication compared to a naive baseline implementation [CB16]. The designed kernel is 3.4× faster than cuBLAS and the MLP runs 7× faster using the XNOR-POPCOUNT kernel compared to the baseline. Big companies like Xilinx, Intel, and Apple are investing heavily in BNNs due to these advantages [UFG+17, KCS+20, Tec20]. Owing to their low memory footprint and lightweight nature of operations, BNNs are considered attractive for edge applications such as FPGA-accelerators [FZS+19], cryptographic neural network inference systems [LWYY20], and for designing low-bitwidth ConvNets [ZWN+16], among many other applications.

Despite their low-bitwidth operations, the accuracy obtained by BNNs is comparable to that obtained by full-precision neural networks. For instance, [ZWN+16] found the accuracy loss of a BNN-ConvNet with 1-bit weights and activations to be less than 0.5% compared to a full-precision ConvNet with seven convolutional layers and one dense layer, evaluated on the Google Street View House Number (SVHN) dataset. Similarly, [LZP17] achieved less than 5% accuracy loss on the ImageNet dataset [DDS+09], a challenging dataset notorious for its complexity in the computer vision community, using a ResNet-like network built entirely using binary convolution blocks, compared to a full precision network.

BNNs apply constraints on both weights and activations such that each element of these matrices is either $+1$ or -1, during both training and at run-time. This is achieved by applying a binarization function, either deterministic or stochastic, to both weights and activations before they are utilized in linear layer arithmetic. In this work, we utilize the deterministic binarization function, `sgn(x)`, to facilitate our discussions due to its simplicity of implementation in hardware and comparable accuracy to its stochastic counterpart. Following equation describes the deterministic binarization function.

$$\texttt{sgn(x)} = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{1}$$

While forward pass uses binarized weights and activations, the gradients computed during training are real-valued to allow Stochastic Gradient Descent (SGD) to calibrate the weights in small, iterative steps. Gradient Descent algorithm tries to find the minimum of the loss function using its gradient. The algorithm starts at an arbitrary point of the



X W Y $g(Y)$ $BIN(g(Y))$
Flatten
x_0 x_1 x_m
$w_{0,0}$ $w_{0,1}$ $w_{m,n}$
b_0 b_1 b_n
$y_0^{\mu_0,\sigma_0,\gamma_0,\beta_0}$ $y_1^{\mu_1,\sigma_1,\gamma_1,\beta_1}$ $y_n^{\mu_n,\sigma_n,\gamma_n,\beta_n}$
Repeat for l layers
Predictions Labels
0 1 2 3 4 5 6 7 8 9
Linear Layer
$y = x \cdot W + b$
BN
$g(y) = \gamma \frac{y-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta$
BIN
$BIN(g(y)) = sgn(g(y))$
Dense Layer

Figure 2: Baseline BNN-MLP dense layer architecture. Note that the last layer does not use the activation function to generate continuous score values.

objective function and computes its gradient with respect to each feature. The parameter value is updated in small steps according to the calculated gradient values at each step, leading to zero gradient. SGD is utilized to reduce the computations that baseline gradient descent needs for calculating the derivatives. The baseline gradient descent algorithm needs to compute the derivatives with respect to each feature for all data points. For large datasets, this is infeasible and makes the gradient descent algorithm too slow. Hence, SGD solves this problem by randomly picking only one data-point from the dataset to compute the gradients, therefore 'stochastic'. Although this significantly reduces the total number of computations to reach the minima, the path taken to reach this minima becomes more noisy due to the random data-points. But this does not matter as long as the algorithm does converge to the minima in lesser training time.

While the binarized activation function `sgn(x)` is simple to implement, its gradient is zero for almost all values of x. Hence, the loss function gradient with respect to the activation function output will be zero, and the network will not learn. To counter this issue, Bengio *et al.* proposed a straight-through estimator for propagating gradients through the neurons [BLC13] as

$$g_x = g_q \mathbf{1}_{|x| \leq 1},$$

where $g_x = \frac{\partial L}{\partial x}$ denotes the gradient of loss function with respect to x. A straight-through estimator sets the input gradients equal to the output gradients, discounting the slope of the $sgn(x)$ activation function altogether [CB16]. Simultaneously, it constrains the real-valued weights to $[-1, +1]$ after each weight update during training since large magnitudes of weight values significantly worsen the performance. High-granularity input features are needed in the first layer in order for the network to be able to distinguish between them. Hence 8-bit input features are utilized for the first layer, while the rest of the layers use binarized features.

Baseline BNNs with deterministic activation function achieve 99.04% and 88.60% accuracy on MNIST and CIFAR-10, respectively. In comparison, [BIL+15] achieved 98.65% accuracy on the MNIST dataset using both binary weights and activations during training. Using only binary weights and full precision activations, [CBD15] achieved an improvement of only 1.5% accuracy on CIFAR10 and reduction of 0.33% on MNIST. These examples demonstrate the comparable accuracy of BNNs despite their significantly lower memory and compute footprints compared to higher precision networks. Figure 2 shows the dense layer architecture of a baseline BNN-MLP. The architecture is similar to that of a generic MLP where a linear layer is followed by normalization followed by an activation function. However, BNN dense layers utilize binary $\mathbf{x}$ and $\mathbf{W}$ and an `sgn(.)` activation function.

The Sign Bit Leakage Problem in Masked BNNs. Regular BNNs with integer additions leak via the sign bit in arithmetic masking because the output sharings are non-uniform [DCA20b]. The discussed vulnerability does not happen in modular arithmetic as *there is no sign bit; the modulo operation wraps around the result if it is out of bounds.* We propose a radically different approach at the algorithmic level that changes the arithmetic of neural networks from integer to modular and defines the inference operations with the new arithmetic from the ground up.

4 ModuloNET: Binary Networks with Modular Arithmetic

The sign-bit leakage problem has motivated us to explore neural network inference in modular arithmetic. Thus, we develop a neural network architecture in which all the integer additions are replaced with modular additions ($\texttt{mod}$ K). To our best knowledge, this is the first paper that utilizes modular arithmetic *directly* during neural network inference[2].

4.1 BNNs and Modular Arithmetic: Challenges

BNN inference has four major issues when layer arithmetic is moved to modular arithmetic.

1. Modulo operation entails loss of information and sudden changes in representations due to overflows.

2. Baseline binary activation function, $\texttt{sgn}(\cdot)$, no longer stays feasible. For feature vector $\mathbf{x}$ where $\mathbf{x} = \{x_m\}$ for $m \in \mathbb{N} \cap [1, M]$, and M is the size of input vector, if x_m lies in a $\texttt{mod}\ K$, then

$$\texttt{sgn}(x_\mathrm{m}) = \texttt{sgn}(K)\ \forall\ m \in \mathbb{N} \cap [1, M] \tag{2}$$

   Hence, activations of all layers are tied to a constant, $\texttt{sgn}(K)$.

3. Application of BN becomes challenging. Modulo operation shifts the negative activations to the positive side (see Figure 3). Thus, BN cannot normalize the mean and standard deviation of the activations to a Gaussian distribution centered around zero anymore.

4. Modulo folding causes negative output scores of the last layer to wrap around. This results in wrong predictions to have high confidence scores (see Figure 3 (c)-(d)).

We analyze these issues and propose modifications to the baseline binary dense layer architecture that allow successfully operating with modular arithmetic in inference.

In supervised representation learning with regular arithmetic, the network automatically discovers its internal features during the training process. The operations of the neural

[2]Homomorphic encryption too uses modular arithmetic but does not *directly* apply such computations; instead it converts regular operations to homomorphic ones—Section 10 elaborates on the similarities and differences of our approach compared to the homomorphic encryption.

network are unconstrained in terms of representation range. However, constraining the inputs of the activation function in a feed-forward network to a modular domain during inference reduces the accuracy of the network since the modulo operation results in loss of information as highlighted in challenge 1. Figure 3 (a) shows the histogram distribution of output features, $\mathtt{g}(\mathbf{y})$, of the third layer of a baseline binarized MLP trained for MNIST. Figure 3 (b) shows the histogram distribution of output features of the same layer when the output of the dense layer is restricted to $\mathtt{mod}\ K$ where $K = 4096$. Intuitively, reducing this representation space increases the information loss due to overflows and underflows of the feature representations with modular arithmetic.

Furthermore, in an unsigned finite domain $K > 0$, the network loses the capacity to represent negative activations, which are fundamental to the functionality of BNNs, highlighted in challenge 2. Equation (2) illustrates that the usual activation function `sgn(.)` cannot work with $\mathtt{mod}\ K$ and requires modifications. Applying the constraint of modular arithmetic also reduces the effectiveness of BN (challenge 3) since modular arithmetic causes sudden changes in values for some activations, while leaving others completely unchanged, and BN operation does not accommodate these anomalies.

In the last layer, the negative scores output from the network wrap around because they underflow the representation range as identified in challenge 4 and illustrated in Figure 3 (c) and (d). This results in wrong predictions with high confidence because baseline neural networks use higher scores to represent high confidence predictions. Plot (c) shows the final layer output of a baseline BNN trained on MNIST. Value bins with high positive values represent correct classifications with high confidence, while those with high negative scores represent the wrong classifications with high confidence. Plot (d) shows the final layer output after applying $\mathtt{mod}\ 4096$: high confidence wrong classifications are folded over and incorrectly classified by the network as correct with high confidence.

4.2 Proposed Layer Architecture

We propose modifications to the baseline dense layer architecture, whose output empirically follows the output of dense layers in baseline BNNs, however, for offline inference, the layer operations are constrained to modular arithmetic. To achieve this, BN can be fused into the preceding linear layer using a simple composition [YN17]. But to maintain binary XNOR-based multiplications between the feature vector $\mathbf{x}$ and weight matrix $\mathbf{W}$, and integer additions of bias vector $\mathbf{b}$, the proposed architecture needs to ensure that the re-parametrization of learned parameters does not lead to non-binary weights and non-integer biases. Specifically, for the linear operation $\mathbf{y} = \mathbf{x} \cdot \mathbf{W} + \mathbf{b}$, the layer needs to ensure that $\mathbf{W}$ and $\mathbf{b}$ tensors maintain their data-types (i.e., binary and integer) during the re-parametrization to allow efficient hardware implementation and masking.

Baseline BNNs use integer biases in their linear layer but since we re-parametrize the layer to fuse the BN operation, the bias values are modified with the BN parameters. We use vanilla BN: $\mathtt{g}(\mathbf{y}) = \boldsymbol{\gamma}\frac{\mathbf{y}-\boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}} + \boldsymbol{\beta}$ in our experiments which generates floating point results during training. $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$, $\boldsymbol{\gamma}$, $\boldsymbol{\beta}$, are the mean, standard deviation, scale, and shift parameter vectors of the BN operation, respectively, ϵ is a small constant to prevent division by zero, and $\mathbf{y}$ is the linear layer output vector. We can absorb these learned parameters offline to obtain a more concise and linear, multiply-accumulate based BN operation as $\mathtt{g}(\mathbf{y}) = \boldsymbol{\psi}\mathbf{y} + \boldsymbol{\phi}$ where $\boldsymbol{\psi} = \frac{\boldsymbol{\gamma}}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}$ and $\boldsymbol{\phi} = \boldsymbol{\beta} - \boldsymbol{\psi}\boldsymbol{\mu}$ are the absorbed BN parameter vectors, computed offline after training. We propose a $\mathtt{mod}\ K$-aware binary activation function based on the shape of the feature map distribution input to the function:

$$\mathtt{BIN}(z) = -\mathtt{sgn}\left(z - \frac{K}{2}\right) \tag{3}$$

This activation function shifts the activation decision boundary from $z = 0$ in vanilla

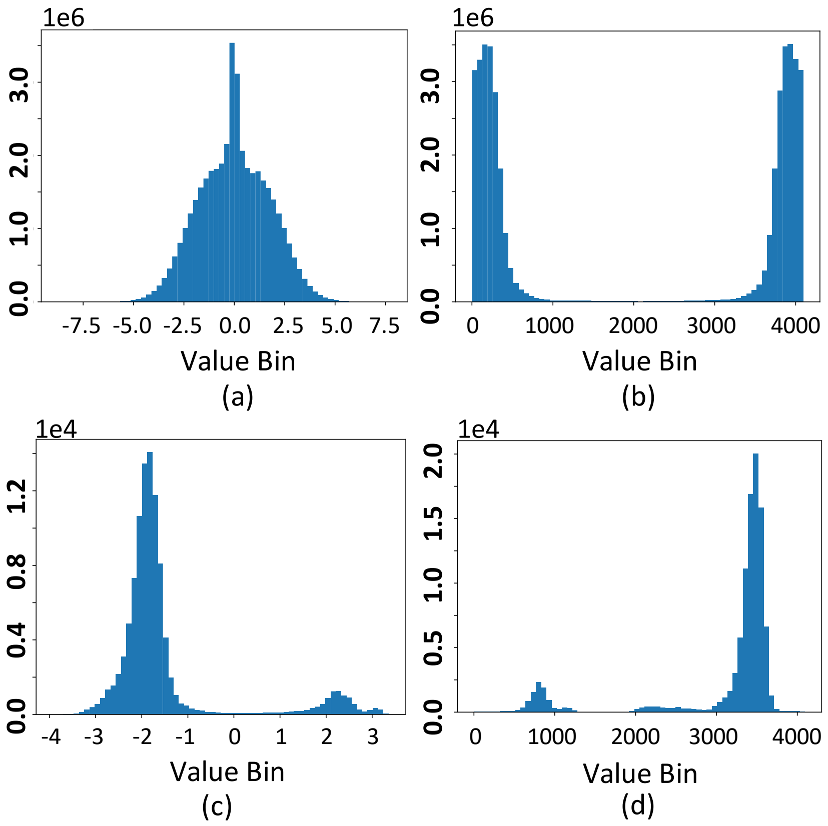


Figure 3: The histogram of feature maps before binarization for (a) third hidden layer, (c) output layer, for baseline MLP, and (b) third hidden layer, and (d) output layer, for ModuloNET MLP, using 64 bins. Activations are obtained using feed-forward run of the networks on the entire MNIST test split. Vertical axis shows the number of occurrences of activation values in the bin while horizontal axis shows the histogram bins.

BNNs to $z = \frac{K}{2}$, the center of the distribution in the $\texttt{mod}\ K$. The negative sign undoes the impact of modulo on the negative outputs of the $\texttt{BN}(\texttt{Linear}(\cdot))$ composition, which causes all output feature values $g(\mathbf{y}) < 0$ to fold, where $K > 0$. Empirically, our activation function allows the layer to 'undo' the modulo-introduced folding and generate an output activation distribution similar in shape to that obtained in baseline BNNs.

To fuse the BN into the preceding linear layer by composition, we note that the input to the activation function lies in $\texttt{mod}\ K$: $g(\mathbf{y})\ \texttt{mod}\ K = (\boldsymbol{\psi}\mathbf{y} + \boldsymbol{\phi})\ \texttt{mod}\ K$.

Our proposed activation function from Equation (3) can be re-formulated as

$$\texttt{BIN}(z) = \begin{cases} 1, & \text{if } z \leq \frac{K}{2} \\ -1, & \text{otherwise} \end{cases}$$

Consider the case when the output of this activation is +1, i.e., $\texttt{BIN}(g(y_n)\ \texttt{mod}\ K) = +1$,

$$\Rightarrow (\psi_n y_n + \phi_n)\ \texttt{mod}\ K \leq \frac{K}{2} \tag{4}$$

The network applies the following constraint on the value of K

$$K \geq 2\ \texttt{max}(|\psi_n y_n + \phi_n| : n \in \mathbb{N} \cap [1, N]), \tag{5}$$

and ensures that the modulo operation is lossless. It can effectively reconstruct the unmodulated input $\psi_n y_n + \phi_n$ after batch-normalization that allows it to later undo the modulo folding. Thus, although modulo operation causes loss of information in general, original signal can be reconstructed with high probability if K is chosen to be large such that the operation does not cause values to scramble after folding. Formal proof is given as follows.

Lemma 1. *n-th signal element, $\psi_n y_n+\phi_n$, can be reconstructed from $(\psi_n y_n+\phi_n)$ mod K $\forall\ n \in \mathbb{N} \cap [1, N]$ with high probability conditioned on occurrence*

$$O_n \triangleq \left\{\psi_n y_n + \phi_n \in \left[-\frac{K}{2}, \frac{K}{2}\right)\right\} \tag{6}$$

Proof. Let $z_n = (\psi_n y_n + \phi_n)\ \texttt{mod}\ K$. Also let

$$\tilde{z}_n = \left(z_n + \frac{K}{2}\right)\ \texttt{mod}\ K - \frac{K}{2} \tag{7}$$

As probability $Pr(O_n) \to 1$, it can be seen from (7) that $\tilde{z}_n \to \psi_n y_n+\phi_n$, which corresponds to approximate regeneration of the original signal element. Furthermore, from Equation (6), $Pr(O_n) = Pr(|\psi_n y_n + \phi_n| \le \frac{K}{2})$. Considering the case of totally lossless modulo folding of the n-th signal element, i.e., $Pr(O_n) = 1$, this is only possible if $|\psi_n y_n + \phi_n| \le \frac{K}{2}$.

Extending this to lossless regeneration of all N signal elements, that is

$$Pr(O_1) = 1, Pr(O_2) = 1, ..., Pr(O_N) = 1 \tag{8}$$

We know that if $Pr(O_i) = 1$ for $i = n$ which produces $\texttt{max}(|\tilde{z}_n| : n \in \mathbb{N} \cap [1, N])$, then in order for Equation (8) to be true, the following must hold

$$\texttt{max}\left(|\psi_n y_n + \phi_n| : n \in \mathbb{N} \cap [1, N]\right) \le \frac{K}{2}$$

This is the same result as Equation (5) which corresponds to lossless modulo folding of the signal. Hence, original signal elements, $\psi_n y_n + \phi_n$, can be unwrapped from the modulo-folded signal elements, z_n, using Equation (7), under the condition that K is selected using Equation (5). □

Hence, under the constraint enforced by Equation (5) and for the case of $\texttt{BIN}(g) = +1$, we can rewrite Equation (4) as $\psi_n y_n + \phi_n \ge 0$. We modify the training process to apply the constraint $\texttt{Clip}(\boldsymbol{\gamma})$ to be between 0 and $+\infty$. This enforces the $\psi > 0$ constraint on Equation (4) and results in

$$y_n \ge -\frac{\phi_n}{\psi_n}$$

$$\sum_{i=1}^{M} x_i w_{i,n} + b_n \ge -\frac{\phi_n}{\psi_n}$$

$$\sum_{i=1}^{M} x_i w_{i,n} + b_n + \frac{\phi_n}{\psi_n} \ge 0$$

A similar analysis can be done for $\texttt{BIN}(z) = -1$ to show that when the activation output is -1, then, $\sum_{i=1}^{M} x_i w_{i,n} + b_n + \frac{\phi_n}{\psi_n} < 0$. Hence, for the case of $\texttt{BIN}(z) = +1$, the value $\sum_{i=1}^{M} x_i w_{i,n} + b_n + \frac{\phi_n}{\psi_n} \ge 0$, and less than 0 otherwise. In $\texttt{mod}\ K$, the activation function gets the composition `BN(Linear(.))` as an input, given by $\left(\mathbf{x} \cdot \mathbf{W} + \mathbf{b} + \frac{\boldsymbol{\phi}}{\boldsymbol{\psi}}\right)\ \texttt{mod}\ K$.

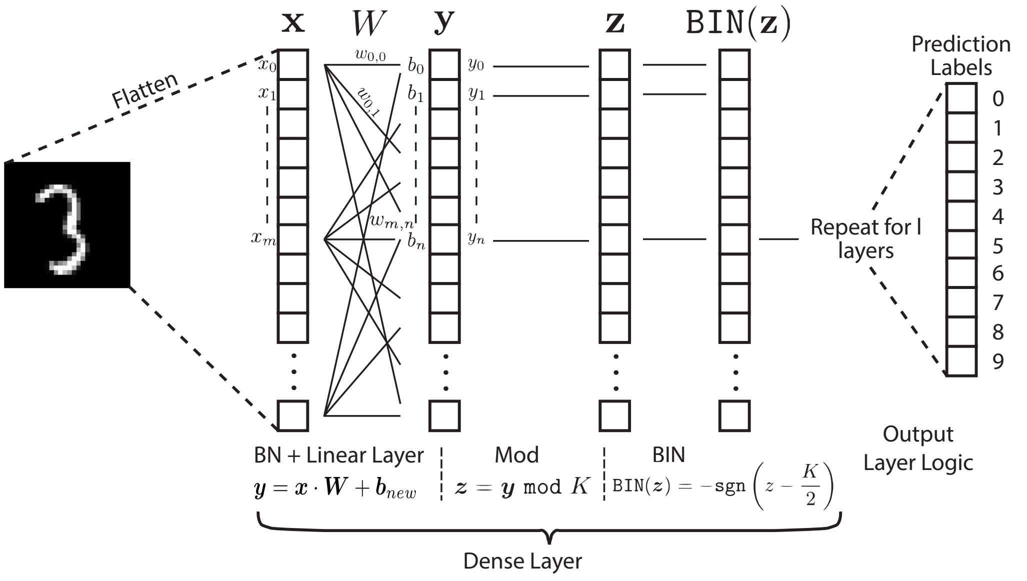


Figure 4: ModuloNET dense layer architecture for inference. The output of the last layer is processed using a more complex logic given by Algorithm 1.

Given both $\mathbf{b}$ and $\boldsymbol{\phi}, \boldsymbol{\psi}$ vectors are available offline after training, we re-parametrize bias vector as $\mathbf{b}_{new} = \texttt{round}(\mathbf{b} + \frac{\boldsymbol{\phi}}{\boldsymbol{\psi}})$. This means that BN operation is absorbed into the preceding linear layer, using re-parametrization of the bias vector $\mathbf{b}$. Round operation $\texttt{round}(\cdot)$ allows $\mathbb{R} \to \mathbb{Z}$ mapping of biases, enabling integer addition of bias vector in hardware. Since $\mathbf{W}$ matrix does not change as a result of the re-parametrization, we can still use XNOR-multiplication in hardware. Layer architecture for inference with fused BN is shown in Figure 4.

4.3 A Note on Enforcement of Constraints

We applied two constraints when designing the modified dense layer architecture for ModuloNET, 1) Constraint on size as $K \geq 2\ \texttt{max}(|\psi_n y_n + \phi_n| : n \in \mathbb{N} \cap [1, N])$, and 2) Constraint on BN parameter $\boldsymbol{\gamma}$ during training, $\texttt{Clip}(\boldsymbol{\gamma}, \mathbf{0}, +\boldsymbol{\infty})$. The empirical evaluation of these constraint's impact is as follows.

For the first constraint, note that choosing the modulus K becomes a challenge as the size is dependent on the components of the feature vector $\mathbf{y}$. Choosing an arbitrarily large K solves the issue but is sub-optimal given size K directly translates to the amount of randomness needed for hardware masking that we discuss in the Sections 6 and 7. However, $\mathbf{y}$ is implicitly batch-normalized by the linear layer due to the re-parametrization of bias vector $\mathbf{b}_{new}$ meaning smaller arbitrary K could be chosen, but the process of choosing K arbitrarily is not scalable for networks with large number of layers. For our network, we choose K based on the range of $\mathbf{y}$ distribution of an infinite domain network, $\texttt{mod}\ \infty$, allowing us to find the smallest K values of each layer, leading to a more optimal design. This requires training the baseline BNN from scratch in infinite domain to extract the $\mathbf{y}$ distributions. Future works could explore the possibility of making K a trainable parameter, removing the explicit dependency of the size on $\mathbf{y}$.

For the second constraint, representation learning handles $\boldsymbol{\gamma}$ well, as the rest of the BN parameters are unconstrained. Based on the fact that the accuracy results of our network with constrained $\boldsymbol{\gamma}$ are found to be similar to those without the constraint, we predict that the network negates the constraint on $\boldsymbol{\psi}$ using the freedom of calibration of the rest of the

Algorithm 1 Output Layer Functionality in `ModuloNET`

```
1: procedure CLASSIFY($\mathbf{z}, N, label$)
input: $\mathbf{z} = z_0, z_1, z_2, ..., z_{N-1}$ $N = num\ classes$
output: $predicted\ label$
2:      $label\_local \leftarrow -1$
3:      $label\_global \leftarrow -1$
4:      $local\_max \leftarrow -1$
5:      $global\_max \leftarrow -1$
6:      for $i = 0 \cdots N-1$ do
7:          if $(z_i > local\_max)$ then
8:              if $z_i < \frac{K}{2}$ then
9:                  $local\_max \leftarrow z_i$
10:                 $label\_local \leftarrow i$
11:         if $(z_i > global\_max)$ then
12:             $global\_max \leftarrow z_i$
13:             $label\_global \leftarrow i$
14:     if $local\_max \neq -1$ then
15:         return $label\_local$
16:     else
17:         return $label\_global$
```

parameters such as those in $\boldsymbol{\phi}$, allowing the network to perform just as well as without this constraint.

4.4 Tuning First and Last Layers

Baseline BNN uses non-binary feature map inputs because a higher granularity of input feature map data is crucial for the network to learn appropriate representations. It also normalizes the 8-bit unsigned input pixels to floating point values in the range $[-1, +1]$. However, our proposed architecture with modular arithmetic casts the 8-bit unsigned integers to a signed representation in the range $[-128, 127]$. Furthermore, it utilizes a larger K for the first layer since non-binary feature map inputs increase the size required to ensure lossless modulo folding.

Given the fact that the modulo operation makes the domain unsigned for $K > 0$, the output of the final layer has to be processed differently than the conventional `max`(scores) approach to map the scores to the correct labels. Figure 3 (c) and (d) show that under the constraint $K \geq 2\ \texttt{max}(|\psi_n y_n + \phi_n| : n \in \mathbb{N} \cap [1, N])$, all activations that would be less than 0 in an infinite domain BNN reside between $\frac{K}{2}$ and K in the `mod` K domain due to folding. Hence, to recover the correct score output of the final layer, the network utilizes a more complex logic to find the score-to-output mapping. Our output layer algorithm to find the correct class, given the last layer node scores, is given in Algorithm 1. This algorithm finds the maximum score below the $\frac{K}{2}$ threshold, called *local_max*, and the global maximum score, called *global_max*. Label corresponding to *local_max* is returned if this max exists, otherwise, the global label is returned.

4.5 Choice of modulo K

As detailed in Section 4.2, ModuloNET dense layer architecture depends on careful selection of an additional hyper-parameter, the value of modulo K. The constraint on K dictated by Equation (5) is needed to ensure lossless folding of modulo since if K is chosen to be too small, modulo folding would cause values to scramble leading to loss of data. The value of K is dependent on the size of data input to the layers of the neural network. BNNs utilize binary values of activations and weights in all layer computations, with the exception of

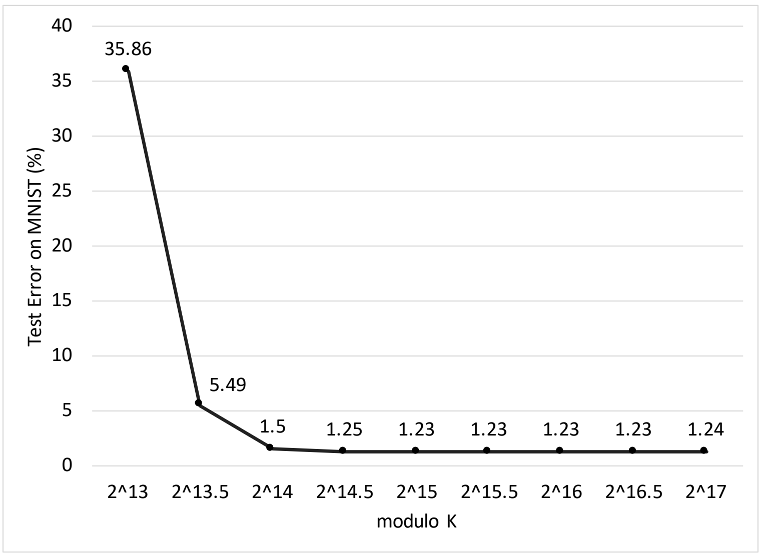


Figure 5: Error rate of MLP-MNIST vs value of modulo K used in the network. Each point represents one test configuration using the specified K in all layers (Decimal exponents are shown only for highlighting the error trend in software implementation, actual hardware implementation only uses integer exponents).

the first layer. The exception comes from the fact that the learned weights need to encode high bit-width information in at least the first layer in order for the network to learn the fine-grained pixel information [CB16]. Hence, a larger K is required for the first layer. Since our goal is to design a hardware block which is reusable for all the layers of the neural network, we select the same K value for all layers. Furthermore, since K denotes the data representation bit-width, we select K to be an exponent of 2. Figure. 5 shows the variation of error rate of ModuloNET with values of K on the MNIST dataset. The error rate becomes roughly asymptotic around $K = 2^{14}$, while achieving the lowest error rate at $K = 2^{15}$. Hence, we utilize $K = 2^{15}$ in our hardware design as 15-bits are required to achieve the minimum error rate. The results in Figure. 5 are intuitive: lower the mod K, higher the likelihood of modulo folding causing values to scramble; the folding no longer stays lossless.

4.6 Extension to ConvNets

ConvNet layers utilize similar operations as those in MLP, with the added maxpool operation in some of the layers. While maxpool appears between the convolutional (conv) layer and the BN operation, to facilitate the fusion of BN into the conv layer biases, BN operation should directly follow the conv layer. We note that `maxpool(BN(.))` composition is permutation-invariant. Hence, pooling operation commutes with BN under the constraint $\boldsymbol{\gamma} > \mathbf{0}$. Note from Section. 4.2 that we apply this constraint anyways using $\texttt{Clip}(\boldsymbol{\gamma}, \mathbf{0}, +\infty)$ for fusion of BN, and that it does not impact the network accuracy due to the freedom of calibration of the rest of the BN parameters. Hence, the network swaps the positions of BN and maxpool to allow BN to appear after conv. Sub-sampling using maxpool before applying BN is beneficial for faster training, however, the fusion of BN into the preceding conv layer keeps the inference complexity unaffected.

We also move the maxpool operation after the binarized activation function to maintain consistency with the layer structure used in our MLP. In our network, maxpool is applied to binary pooling windows, returning +1 if any element of the window is +1 and generating

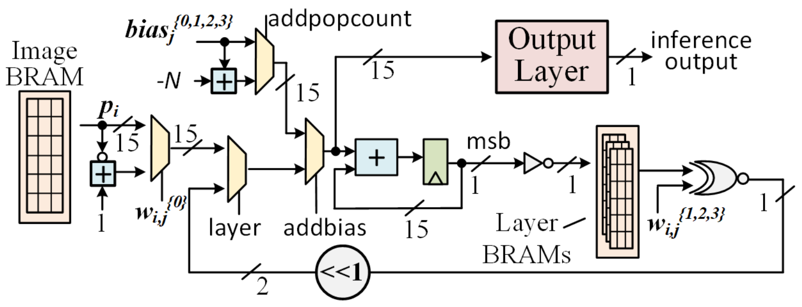


Figure 6: Hardware design of unmasked ModuloNET inference. The design accumulates the partial products in each cycle. It stores the inverted MSBs of the complete summation as the activations. For the output layer, it directly feeds the final summations to the output layer logic that computes the inferred label and its confidence score.

-1 if all components are -1. The re-parametrization of bias vector for fusion of BN is similar to that in the MLP, i.e., $\mathbf{b}_{new} = \texttt{round}(\mathbf{b} + \frac{\phi}{\psi})$.

5 Baseline Hardware Design of ModuloNET

To obtain baseline hardware for our side-channel and overhead comparisons, we first design and implement an unmasked design of ModuloNET inference. It is an area-optimized, sequentialized hardware design for the MLP that performs one summation per cycle[3]. This allows the design to fit in our target FPGA and a direct comparison with earlier work [DCA20a]. The design has a throughput of 1 addition per cycle. For the following discussions, we denote the number of input pixels, hidden layer nodes, output layer nodes, and the modulus with L, M, N, and K, respectively. The design is parameterized for these values. The weights and activations are binary (i.e., -1 and +1 are represented as 0 and 1 respectively) and the bias value per node has a precision of $log_2 K$ bits. $K = 2^{15} = 32768$ achieves the lowest error rate for our datasets. Thus, unless otherwise stated, all the weighted summations are performed in modulo 32768 and we omit the 'mod 32768' term.

5.1 Weighted Summations and XNOR-POPCOUNT

Figure 6 shows the hardware implementation of the weighted summations and XNOR-POPCOUNT. The image pixels and network parameters are loaded into a block RAM at the start of each inference. The hardware reads out the i^{th} input pixel p_i and the weight $w_{i,j}^{\{0\}}$ that corresponds to the connection between the i^{th} input pixel and j^{th} hidden layer node (see Figure 4) from the respective block RAMs. It uses a multiplexer with binarized weight $w_{i,j}^{\{0\}}$ as its select line, and the pixel p_i and its 2's complement $(-p_i)$ as its data lines for the multiplication. It feeds the product to an accumulator. To perform the additions in $mod\ K$, we restrict the width of the accumulator data path to $log_2 K$ bits. The hardware feeds the sum of L partial products and $bias_j^{\{0\}}$ to the activation function after $L+1$ cycles and saves the activation function outputs into layer BRAMs. For subsequent layer computations, the hardware reads the activation values $a_i^{\{k\}}$ and weights $w_{i,j}^{\{k\}}$ from the k^{th} layer and weight block RAMs and feeds them to the XNOR-POPCOUNT (`XP`) function defined in Section 3.3. XNOR is directly implemented on hardware using XNOR gates. To implement POPCOUNT, the hardware uses a left-shift operator to perform the

[3]One neuron computation with L input connections and a bias finishes in $L+1$ cycles.

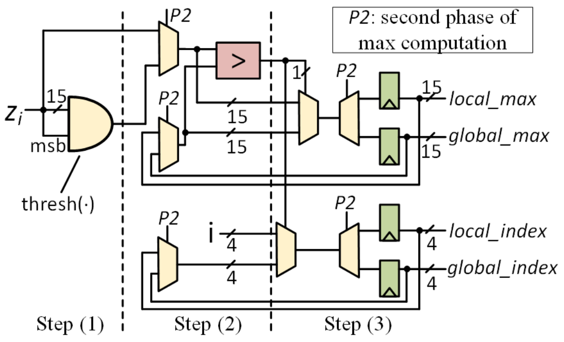


Figure 7: Hardware design of the unmasked output layer. z_i represents confidence score of the i^{th} node in the output layer.

multiplication by 2 and then adds $-M$ along with $bias_j^{\{k\}}$ in the last cycle of weighted summation for the j^{th} hidden layer node.

5.2 Activation Function

The activation function in ModuloNET is given in Equation (3). To implement it on hardware, we reformulate the function $-\mathtt{sgn}(x - \frac{K}{2})$, where x represents the weighted sum, to a comparison of x with $\frac{K}{2}$. The hardware checks the MSB[4] of x to compare it with $\frac{K}{2}$. The final output is the inverse of MSB because of the negative sign in Equation (3).

5.3 Output Layer Max Function

Algorithm 1 presents the method to compute the final classification result and Figure 7 shows its hardware implementation. The hardware sequentially feeds each of the N confidence scores (z_i) to the output layer logic. The entire computation of the output layer happens in two phases corresponding to the *local_max* and *global_max* computations. First phase has three steps: (1) feed z_i to a comparator if it is below $\frac{K}{2}$, else feed a 0, (2) compare z_i with the value in *local_max* register [5], and (3) update the *local_max* register with z_i and the *local_index* register with i if the comparator returns 1. Next, we sequentially describe the baseline hardware implementation of each step.

1. The hardware should return z_i if it is less than $\frac{K}{2}$ (or equivalently if the MSB is equal to 0) and 0 if it is greater than $\frac{K}{2}$ (or equivalently if the MSB is equal to 1). Thus, the function `thresh(.)` can be expressed as an AND of z_i and inverted MSB.

   $$\mathtt{thresh}(z_i) = [z_{[14]}, z_{[13]}, \cdots, z_{[0]}] \odot [\overline{z_{[14]}}, \overline{z_{[14]}}, \cdots, \overline{z_{[14]}}],$$

   where $\odot$ is an element-wise AND operation. The design only requires $log_2 K - 1$ number of AND gates for this function because the MSB of the output is always 0.

2. The hardware feeds the thresholded confidence scores to a $log_2 K$-bit comparator, which compares them with the stored value in *local_max* register.

3. The design uses comparator output as a select for the multiplexer that either selects the value in *local_max* or z_i to update the *local_max* register. The *local_index* register is also accordingly updated.

[4]For any n-bit number, the MSB reveals if the number is greater or less than 2^{n-1}; $K = 2^n$.
[5]Both *local_max* and *global_max* registers are initialized to zero at the start of the computation.

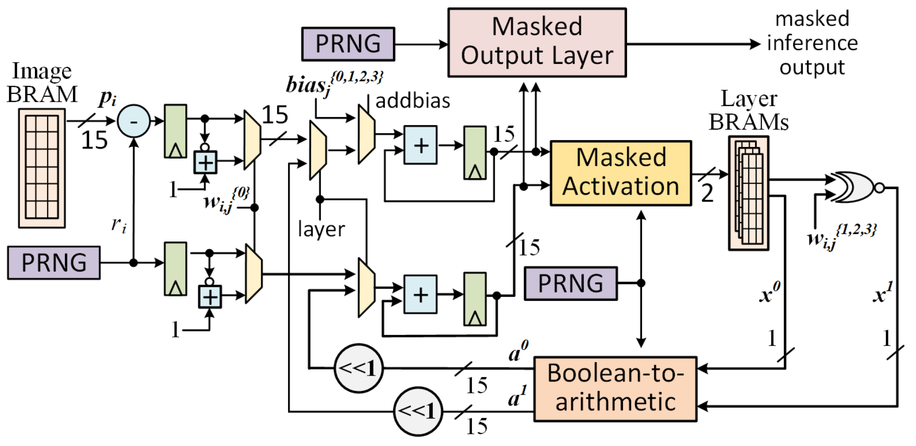


Figure 8: Hardware design of the fully masked ModuloNET inference showing various top-level masked components. The weights $w_{i,j}^{\{k\}}$ and $bias_j^{\{k\}}$ are read from the respective block RAMs. Figure is not to scale.

In the second phase, the hardware directly computes the max of N confidence scores z_i without thresholding. Thus, it directly executes the second and third steps to update the *global_max* and *global_index* registers. The hardware reuses the comparator for both phases by multiplexing the input of the comparator to either select `thresh(`z_i`)` or z_i based on the active phase of max computation handled by the control bit $P2$.

6 Fully Masked Hardware Design of ModuloNET

Figure 8 presents the overall hardware design of the first-order masked ModuloNET inference. The design follows the approach of the unmasked case where the hardware is sequentialized but it processes two shares in parallel. We suitably choose the masking style to be either Boolean or arithmetic based on the nature of operation to be masked for efficiency. We apply DOM with registered outputs for Boolean masking. Next, we describe the masking of all the major operations of the neural network.

6.1 Arithmetic Masking of Weighted Summations

Figure 8 shows the implementation of masked weighted summations using arithmetic masking because it is an arithmetic operation. The hardware reads p_i from the image BRAM, pads it to log_2K-bits, and splits it into two arithmetic shares $p_i - r_i$ and r_i, where r_i is a log_2K-bit fresh random mask. Next, the hardware directly and independently computes the weighted summation on the two arithmetic shares. Finally, the hardware adds $bias_j^{\{0\}}$ to one of the shares and sends both the shares to the masked activation function. The secret weights $w_{i,j}^{\{k\}}$ remain hidden throughout the computation because the hardware combines it with randomized arithmetic shares before starting the computations.

Note that the secret in our model are the weights but we propose to mask the input pixels instead of masking the weights [6]. Applying Boolean masking on the binary weights leads to more area overheads because the primary operation of weighted summation is arithmetic. In terms of security, since the adversary does not know the actual input $(p_i - r_i)$

[6]The process of randomizing the input is commonly known as *Blinding*–the inputs are unchanged but split into two shares. It preserves the functionality—still performs the same classification task.

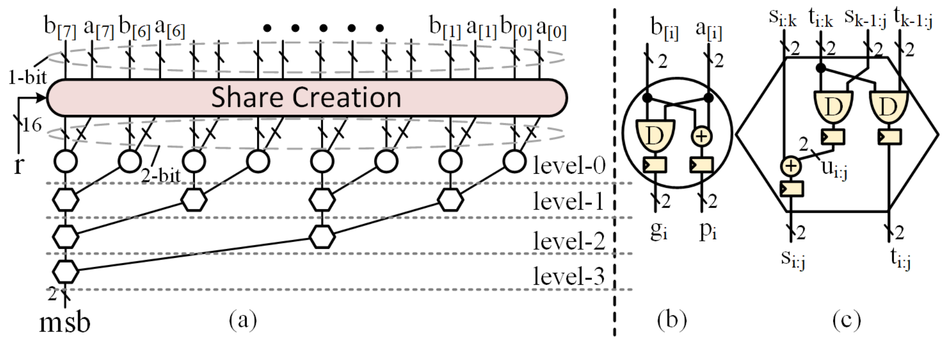


Figure 9: Hardware design of the masked activation function. A logarithmic carry chain propagates the carry using generate and propagate functions. AND gates are replaced with DOM-indep AND labeled as 'D'; we omit the masks in the figure to avoid clutter.

that gets multiplied by the secret ($w_{i,j}$), it cannot create a hypothesis on the intermediate computations for a first-order side-channel attack.

6.2 Masked Activation Function

The activation function receives two arithmetic shares in the masked design and needs to compute if their sum is greater than or less than $\frac{K}{2}$, which depends on the MSB of the sum. Therefore, the hardware only needs to compute the MSB of the final summation in a masked fashion. A prior work solves this problem by implementing a masked LUT-based ripple carry chain to produce the two Boolean shares of the MSB [DCA20b]. However, such a design is good for applications where the security requirements are low; glitches can happen inside the LUTs and eventually reveal the secret [MPO05]. Prior literature suggests a number of alternatives such as Threshold Implementation [NRR06], DOM [GMK16], a synchronized variant of the Trichina's AND gate [DCA20a], etc., to perform masked operations with reduced glitches. We choose DOM for our design because it has low area and randomness requirements and provides a reasonable security against glitches[7].

Figure 9 shows our proposed novel design of the masked activation function using DOM-indep AND gates for 8-bit operands[8]. We specifically use the DOM-indep multiplier instead of DOM-dep because we guarantee independent input sharings: the hardware creates fresh Boolean shares for the arithmetic share bits. The hardware needs to create Boolean shares of the input bits (of arithmetic shares) explicitly before feeding them into the masked AND gates because the logic to compute the carry is completely exposed in a gate-level masked design. The `Share Creation` block creates two Boolean shares for every bit of each arithmetic share.

Instead of adopting a linear ripple-carry design [DCA20b], we implement the carry chain of a Kogge-Stone adder [KS73] because it has a logarithmic latency. We only implement the data paths in the carry chain that are necessary to compute the MSB. We replace the AND gates in the logic with DOM-indep AND gates to mask the design; XORs being linear can directly process the two share domains independently. Level-0 of the tree produces the Boolean shares of carry-generate (g) and carry-propagate (p) terms. Following equations present the masked versions of g and p.

$$(g_i^1, g_i^0) = \texttt{DOM}(a_{[i]}^1, a_{[i]}^0, b_{[i]}^1, b_{[i]}^0, r_a)$$

[7]We discuss some alternative primitives in Section 10.
[8]We show an illustration with 8-bits but the actual design has $log_2 K$-bit operands.

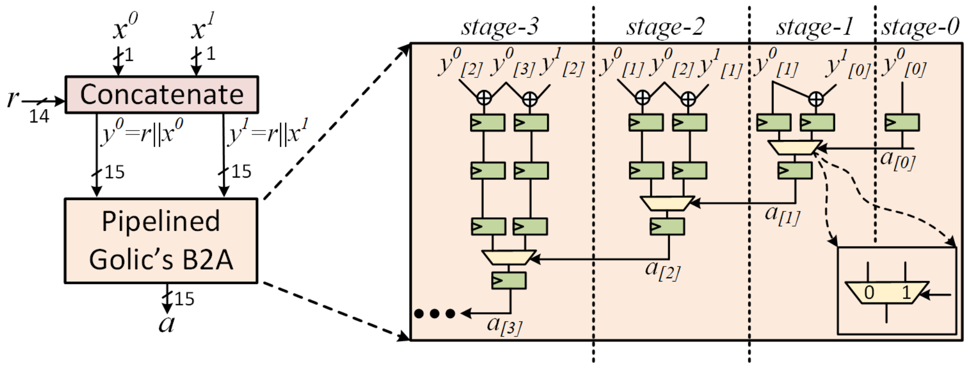


Figure 10: Design of our proposed circuit to process the binary Boolean shares (x^0, x^1) to produce a such that $\bigoplus_i x^i = a + x^1$. The circuit concatenates the Boolean shares with 14 random bits and feeds them to a pipelined version of Golic's Boolean to arithmetic circuit.

$$(p_i^1, p_i^0) = (a_{[i]}^1 \oplus b_{[i]}^1, a_{[i]}^0 \oplus b_{[i]}^0)$$

The subsequent levels process the group generate $(s_{i:k}, s_{k-1:j})$ and group propagate $(t_{i:k}, t_{k-1:j})$ terms to produce the group generate $(s_{i:j})$ and group propagate $(t_{i:j})$ for the next level. For any level $l > 0$, the relation between i and k can be generalized to $i = k + 2^{l-1} - 1$. For level-1, $i = k$ and $s_{i:i} = g_i$ and $t_{i:i} = p_i$. Following equations illustrate the masking of the group generate and group propagate terms.

$$(u^1, u^0) = \texttt{DOM}(t_{i:k}^1, t_{i:k}^0, s_{k-1:j}^1, s_{k-1:j}^0, r_b)$$

$$(s_{i:j}^1, s_{i:j}^0) = (u^1 \oplus s_{i:k}^1, u^0 \oplus s_{i:k}^0),\ (t_{i:j}^1, t_{i:j}^0) = \texttt{DOM}(t_{i:k}^1, t_{i:k}^0, t_{k-1:j}^1, t_{k-1:j}^0, r_c) \quad (9)$$

6.3 Boolean to Arithmetic Conversion

Prior literature shows that care should be taken while converting Boolean shares to arithmetic shares because the process of conversion might leak the original secret [Gou01]. Given the Boolean shares of a secret variable x as x^0 and x^1 such that $\bigoplus_i x^i = x$, the conversion aims to find an element a such that $x^0 \oplus x^1 = a + x^1$, where $+$ is an addition $\texttt{mod}\ K$. We need to design a conversion circuit to convert the 1-bit Boolean shares of the activation to log_2K-bit arithmetic shares.

Figure 10 shows the proposed Boolean to arithmetic conversion circuit. Most prior works on share conversion in hardware assume the same field for inputs and outputs and try to reuse the existing randomness in the Boolean shares [Gol07, MTMM07a]. However, our design does not have log_2K bits in the Boolean shares. Thus, it first converts the 1-bit Boolean shares x^0, x^1 to log_2K-bit Boolean shares by concatenating each share with $log_2K - 1$ fresh random bits. Next, it uses the circuit proposed by Golic to perform a secure Boolean to arithmetic share conversion [Gol07], with full pipelining to resist glitches. Following equations describe how each bit $a_{[i]}$ of the arithmetic shares is produced by processing the bits of the Boolean shares $y_{[i]}^j$, where $0 \le i \le 14$ and $j \in (0, 1)$.

$$a_{[0]} = y_{[0]}^0 \quad (10)$$

$$a_{[1]} = \texttt{MUX}(y_{[1]}^0, y_{[1]}^0 \oplus y_{[0]}^1, a_{[0]}) \quad (11)$$

$$a_{[i]} = \texttt{MUX}(y_{[i-1]}^0 \oplus y_{[i]}^0, y_{[i]}^0 \oplus y_{[i-1]}^1, a_{[i-1]}) \quad (12)$$

The hardware feeds the produced arithmetic shares to the accumulator to process the next layer.

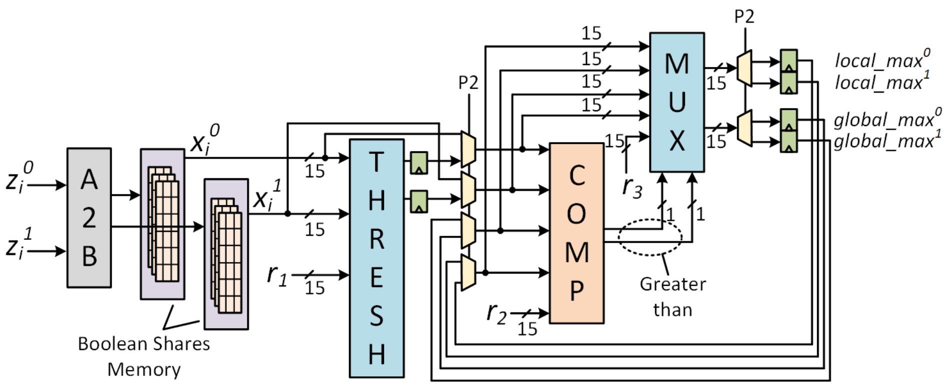


Figure 11: Hardware design of the fully masked output layer. Figure is not to scale.

6.4 Masked Output Layer

In the masked design, the output layer receives the arithmetic shares (z_i^0, z_i^1) of the confidence scores from the two adders and generates the Boolean shared inference result. Section 5.3 describes the three main operations in the output layer for the unmasked design viz. AND, comparison and multiplexer.

Since all the output layer operations like AND, comparison and multiplexing are bit-wise manipulations, we apply Boolean masking to the complete output layer for efficiency. Therefore, the hardware first converts the arithmetic shares to Boolean shares and stores them in a Block RAM. Next, it performs all the operations of the baseline output layer described in Section 5.3 in a masked fashion. Figure 11 shows the hardware design of the fully masked output layer in terms of the four building blocks viz. arithmetic to Boolean converter (`A2B`), masked thresholder (`THRESH`), masked multiplexer (`MUX`) and masked comparator (`COMP`). We discuss the details of these components next.

The Design of Arithmetic to Boolean Share Converter. We again refer to the work by Golic to implement the arithmetic to Boolean share conversion [Gol07]. The work shows that the Boolean equations are symmetric for share conversion—Equations (10), (11), and (12) can be directly used to generate the arithmetic shares by simply swapping the Boolean share bits with the arithmetic share bits. Likewise, the hardware design of the arithmetic to Boolean converter is similar to the design of the *Pipelined Golic's B2A* block shown in Figure 10. However, the hardware feeds arithmetic shares (z_i^0, z_i^1) to this block as Figure 11 shows. The generated Boolean shares (x_i^0, x_i^1) are stored in two separate Block RAMs to be processed in the later steps of the max computation.

The Design of Masked Threshold. The hardware needs to securely threshold Boolean shares (x_i^0, x_i^1) based on whether $x_i^0 \oplus x_i^1$ is greater or less than $\frac{K}{2}$. As mentioned in Section 5.3, this operation can be simplified to an AND operation of the confidence score with its inverted MSB. Similar to the masked activation function, we replace the regular AND gates with masked AND gates in the design that takes in x_i^0, x_i^1, the Boolean shares of the inverted MSB (i.e., one Boolean share inverted) and random mask r_1. Since the MSB of the output of `THRESH` is always zero, the hardware uses a random Boolean sharing of zero for it. Finally, the Boolean shared and thresholded confidence scores are sent to a masked comparator for the next step after registering the outputs.

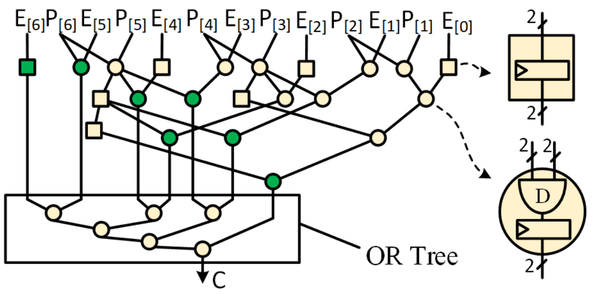

E[6] P[6] E[5] P[5] E[4] P[4] E[3] P[3] E[2] P[2] E[1] P[1] E[0]
2
2
2 2
D
2
OR Tree
C

Figure 12: Hardware design of the masked comparator. The circles and squares on the right represent DOM-indep AND gate and a register. The green circles denote completed partial products of Equation 13 which are fed to the OR Tree for final evaluation of C.

The Design of Boolean Masked Comparator. We first describe a gate-level hardware design of comparator circuit and then explain its masking[9]. The Boolean function `C(.)` of an n-bit comparator is given in Equation (13).

$$\mathtt{C}(a,b) = \mathtt{E}_{[n-1]}|\mathtt{P}_{[n-1]}\cdot\mathtt{E}_{[n-2]}|\mathtt{P}_{[n-1]}\cdot\mathtt{P}_{[n-2]}\cdot\mathtt{E}_{[n-3]}|\cdots|\mathtt{P}_{[n-1]}\cdot\mathtt{P}_{[n-2]}\cdots\mathtt{P}_{[2]}\cdot\mathtt{P}_{[1]}\cdot\mathtt{E}_{[0]}, \quad (13)$$

where $\mathtt{E}_i = a_{[i]}\cdot b_{[i]}$ and $\mathtt{P}_i = a_{[i]}\overline{\oplus}b_{[i]}$ $\forall i \in [0,n)$ represent the bit-wise comparison and equality check, respectively. The expression is amenable to a parallel-prefix computation.

Figure 12 shows the hardware design of a 7-bit comparator as an example. The hardware computes the final comparison in a parallel-prefix fashion by splitting the products with more terms (e.g., $\mathtt{P}_6\cdot\mathtt{P}_5\cdots\mathtt{P}_0$) into multiple products with lesser terms ($\mathtt{P}_6\cdot\mathtt{P}_5$, $\mathtt{P}_4\cdot\mathtt{P}_3$, and $\mathtt{P}_2\cdot\mathtt{P}_1$) and evaluating them in parallel. Subsequently, the hardware combines the partially computed products to fully compute all the product terms in Equation (13) and produce the final comparison result by 'ORing' them. We design its masked version by replacing the regular AND gates with DOM-indep multipliers having registered outputs.

We emphasize that the term 'comparison' in our work refers to *comparing two variable numbers and returning whether one is greater than the other or not.* Other works have also proposed designs for masked comparison, but in those works the term 'comparison' means an equality check [OSPG18]. Furthermore, the operation of thresholding is also a comparison and some works propose its masking [RRd+16], but thresholding compares a variable number to a fixed constant.

The Design of Boolean Masked Multiplexer. Finally, the hardware implements the logic to update the masked max and index registers based on the (Boolean) masked comparator result using a masked multiplexer. From Equation (14), a multiplexer consists of 2 AND and 1 OR operation; the OR can also be written in terms of AND.

$$\mathtt{MUX}(a,b,s) = \overline{\overline{(a\cdot s)}\cdot\overline{(b\cdot\overline{s})}} \quad (14)$$

where $a, b, s \in GF(2)$ are the two data lines and the select line, respectively. To implement the masked version, we simply replace all the regular AND gates with masked AND gates. The masked index computations are performed accordingly.

[9]The gate-level design is not needed for the baseline unmasked design, which uses the default implementation of a comparator by the synthesis tool for the target FPGA. However, masked design needs to define this in Boolean gates and use their secure versions.

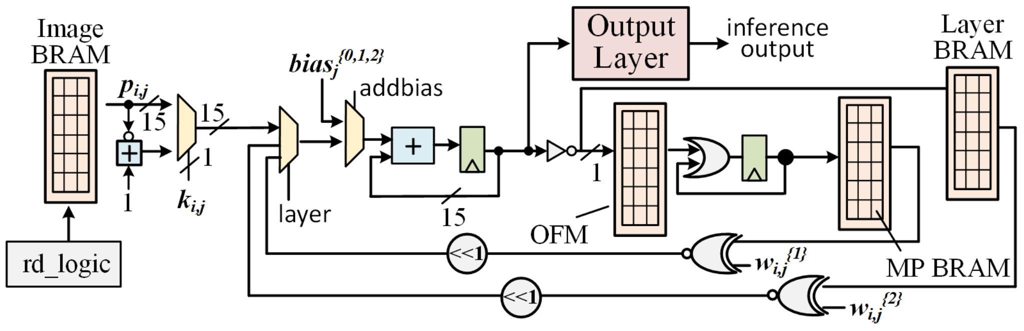


Figure 13: Hardware design of the unprotected binarized ConvNet.

7 Hardware Design of ConvNet

We also implement a ConvNet hardware design to explore the masking of additional operations like convolution and maxpool and quantify the area overheads. Similar to the MLP design, we first implement a sequentialized unprotected ConvNet design and then implement its masked version. We choose a network architecture with 1 conv layer, 1 maxpool layer, 2 FC layers, and an output layer.

We have several reasons for choosing the smaller parameters compared to the original software implementation, which is listed in Section 9.1. First, our goal is to quantify the area overheads of masking; in an area-optimized design, any additional number of layers or node per layer will mostly increase the latency of the design, not the area because most of the design components are reused. Second, a lower latency design reduces the side-channel evaluation time. Third, the complete design will not fit the FPGA that we use. The software results show that our techniques are not limited by the ConvNet hyperparameters and can indeed provide a classification accuracy as good as the baseline. In this section, we demonstrate the techniques to mask the operations in a ConvNet that can be adopted and deployed in any ConvNet with a totally different set of hyperparameters. We discuss the impact of hyperparameters in Section 10.

Unprotected ConvNet Design. Figure 13 shows the hardware design of the unprotected ConvNet. Compared to the baseline MLP design, there are two important differences due to the two new convolution and maxpool operations. First, the ConvNet design requires special read logic for the image pixels and kernel elements: the controller assumes the linearly stored image pixels to be laid out as a 2D matrix, sequentially selects sub-matrices (called convolution windows) within the 2D matrix and generates addresses corresponding to the pixels in the sub-matrix. Second, the design requires logic to perform maxpool operation on the output of the convolutional layer: similar to the convolution operation, the controller needs to generate addresses for maxpool window elements and then select the maximum element in the window. In a BNN, maxpool is equivalent to OR operation and we also use this optimization in our design [UFG+17].

We reuse the weighted summation block of the MLP design to perform convolutions since convolution is also a weighted summation. Using the special read logic, the hardware reads out the pixels $p_{i,j}$ from the Image BRAM (one pixel at a time / clock cycle) and kernel weights $k_{i,j}$ from the kernal BRAM, multiplies them using the multiplexer and feeds the result to the accumulator. At the end of each convolution window, the hardware stores the inverted MSB of the weighted summation as the activated convolution result in output feature map (OFM) BRAM.

After finishing all the convolutions, the hardware initiates the maxpool operation. A

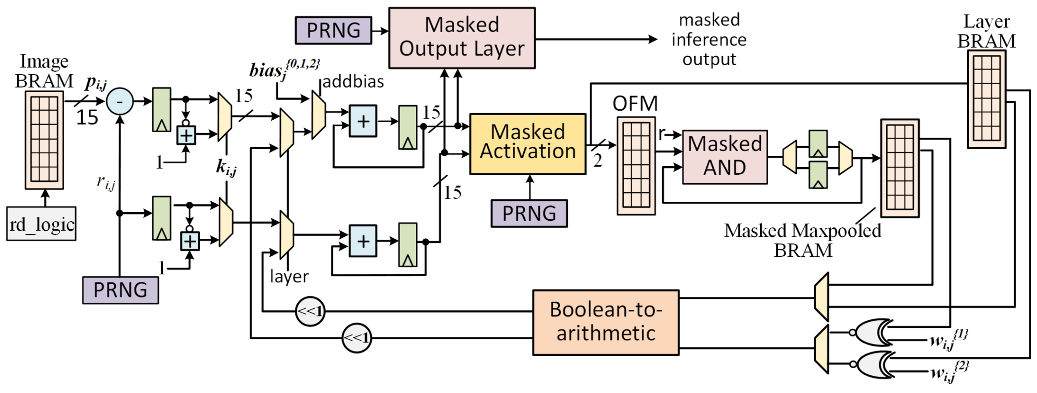


Figure 14: Hardware design of masked ConvNet.

special read logic reads out the elements of the OFM based on the maxpool window selected and feeds them to the maxpool logic. We implement maxpool as an OR accumulator: a register that is initialized to zero keeps storing the OR result of itself with a new input every cycle. The controller resets the register at the start of every maxpool window. Finally, the Maxpooled BRAM (MP BRAM) stores the results.

Since the network only has FC layers after maxpool, the flow is similar to MLP. The hardware processes the maxpooled results sequentially to compute the nodes of the first hidden layer. Finally the hardware computes the output layer confidence scores by processing the activations of the first hidden layer and produces the inference result.

Masked ConvNet Design. In addition to the MLP operations, the ConvNet design has two new operations: convolution and maxpool. The additional control circuits do not pose any security issues in the design because their operation is independent of the values of the secret parameters. However, the convolution and the maxpool operations' datapath need masking.

Figure 14 shows the hardware design of the masked ConvNet. Since convolution is essentially a weighted summation, we use arithmetic masking of the input pixels like we did in the MLP case. The design performs parallel convolutions of the masked inputs $p_{i,j} - r_{i,j}$ and $r_{i,j}$ with the secret kernels weights in two independent datapaths. At the completion of each convolution, the hardware feeds the two arithmetic shares of the convolution result to the masked activation function that we reuse from the MLP design. Finally, the OFM BRAM stores the Boolean shares of the activated convolution result. Maintaining a Boolean sharing assists in masking of the maxpool operation that we discuss next.

Maxpool is an OR operation that can be masked using a masked AND gate, which requires the inputs as Boolean shares. Since the activation function already produces Boolean shares of activation, the hardware does not need a share conversion between convolution and maxpool. The hardware directly reads the Boolean shares from OFM BRAM using the special read logic control and feeds the inputs to the masked AND gate—DOM-indep AND gate in this case. The accumulator stores the output of the masked AND gate and drives the second input of the masked AND gate with the updated result.

However, due to the additional 1-cycle latency of the DOM-indep AND gate, the result from the accumulator only becomes available after 2 cycles for every input. This reduces the original throughput of the design to half, which is not acceptable. We solve this problem by taking inspiration from another prior work on masking of BNNs [DCA20a]. We observe that the maxpool computation of each window is independent of each other. Therefore, we modify the hardware to start the computation of the next maxpool window while the previous result is in-flight. The optimization requires an additional accumulator register to keep track of each maxpool window that it processes concurrently. The hardware stores the Boolean shares of the maxpooled result in Masked Maxpooled BRAM.

The remaining network only consists of FC layers (including the flattened maxpooled outputs that constitute the first FC layer) and works in a similar fashion as the masked MLP design. The masked activation function directly feeds the layer BRAMs of the remaining layers. Finally, we reuse the masked output layer of the MLP in the ConvNet design to produce the masked inference result of the neural network.

8 Security Analysis of the Masked Gadgets

We present the security analysis of our proposed masked hardware gadgets in this section. We first define the two commonly used adversary models in literature viz. *t-probing security* and *glitch-extended t-probing security.*

Definition 1. *t-probing security [ISW03]* A gadget `G` is *t-probing secure*, iff any arbitrary combination of every t-tuple wires in the gadget is independent of any sensitive variable.

Definition 2. *Glitch-extended t-probing security [RBN+15, FGP+18]* A gadget `G` is *glitch-extended t-probing secure*, iff any arbitrary combination of every t-tuple wires in the gadget *and* the wires in their fan-in until the last registered point is independent of any sensitive variable.

Since the focus of this work is to mask ML-specific operations, we prove the first-order security of the ML-specific gadgets in the *glitch-extended probing model* [RBN+15] and provide *1-probing-secure* implementations for other gadgets like `B2A` and `A2B`. Our `B2A` and `A2B` implementations can be replaced with the respective glitch-extended versions from the literature for a stronger security guarantee. For the proofs in the *glitch-extended probing model*, $\mathcal{O}$ denotes *observation set*–the set of all the intermediate nets observable by the adversary. We occasionally use a subscript to distinguish between two sets corresponding to different probe positions. $\mathcal{A}$ can place at most 1 probe in the gadget since since we claim *first-order security.*

We start with proving the glitch-extended probing security of the masked weighted summation gadget that uses arithmetic masking, which is followed by the proofs of other masked gadgets in the design, like the masked activation function, the Boolean-to-arithmetic and arithmetic-to-Boolean converters, masked comparator and the masked multiplexer.

8.1 Weighted Summations:

Figure 15 shows the isolated masked weighted summation gadget `G1`. The circuit computes the summation over masked weighted input pixels during the input layer computations and over masked weighted activation values during the hidden layer computations. Thus, in the input layer, the two inputs to the circuit are the two arithmetic shares $(p_i - r_i) \cdot w_{i,j}^{\{0\}}$ and $r_i \cdot w_{i,j}^{\{0\}}$ of the partial product $p_i \cdot w_{i,j}^{\{0\}}$. In the hidden layer, the gadget inputs are arithmetic shares b^0 and b^1 of the product $a_i \cdot w_{i,j}^{\{k\}}$ of activation value a_i with the respective weight $w_{i,j}^{\{k\}}$ of the k^{th} layer. We aim to protect the weights $w_{i,j}^{\{k\}}$ in this gadget.

Theorem 1. *`G1` is glitch-extended t-probing secure given the secret variables as $w_{i,j}^{\{k\}}$.*

Proof. (Sketch) The gadget is internally split in two independent datapaths $D1$ and $D2$ corresponding to the two share domains. The hardware registers the arithmetic shares before feeding them to `G1`.

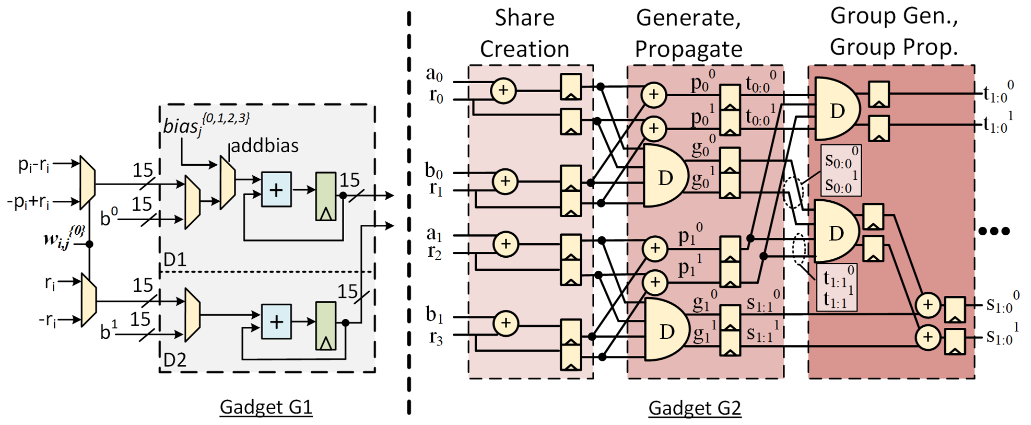


Figure 15: Circuit design of the masked gadgets `G1` and a portion of `G2`. Registers at the end of each sub block guarantee composability of the DOM gadget.

1. During the input layer computations, the input to $D1$ can either be $(p_i - r_i)\ mod\ K$ or $(-p_i + r_i)\ mod\ K$, depending on whether $w_{i,j}^{\{0\}}$ is 1 or 0, respectively. Here, $p_i \in \mathbb{Z}_K$ and is known to $\mathcal{A}$; $r_i \in \mathbb{Z}_K$ is a fresh and uniformly sampled random number; K is the modulus. Thus, for both possible values of $w_{i,j}^{\{0\}}$ the input to $D1$ is always a fresh and uniformly distributed random number. Since the inputs to the gadget are registered, the observation set variables are confined to the intermediate nets inside `G1`. Therefore, any arbitrary function of the intermediate nets in $D1$ will produce only random outputs independent of $w_{i,j}^{\{0\}}$.

2. During input layer computations, the input to $D2$ can either be $r_i\ mod\ K$ or $-r_i\ mod\ K$. Both these values are also fresh and uniformly sampled random numbers. Thus, any arbitrary function of the intermediate nets will also produce outputs independent of $w_{i,j}^{\{0\}}$.

3. For the hidden layer computations, the inputs to $D1$ and $D2$ are the registered outputs b^0 and b^1 from the Boolean-to-arithmetic converter. We prove in Section 8.3 that the outputs from the Boolean-to-arithmetic unit are also fresh and uniformly distributed random numbers. Thus, using a similar analysis as that for the input layer, any arbitrary function of the intermediate nets produced during the hidden layer computations in either $D1$ and $D2$ are independent of $w_{i,j}^{\{k\}}$.

Important Notes. We assume that the encoder circuit that generates the shares of $p_i \cdot w_{i,j}^{\{0\}}$ by loading p_i, r_i and $w_{i,j}^{\{0\}}$ *cannot* be probed by $\mathcal{A}$. Such assumptions on the encoder are common in prior works on provably-secure hardware masking [ISW03]. Furthermore, although the weights are unmasked in the gadget, that is only an issue with template attacks, not DPA.

□

8.2 Masked Activation Function:

We refer to this function as gadget `G2`. Figure 9 shows the abstract-level diagram of the full design and Figure 15 shows a detailed circuit of a small portion of the design that processes the bits a_0, a_1, b_0, and b_1 until level-1 of the tree. It takes as input 15-bit arithmetic shares S^0, S^1 of the actual weighted summation S and produces Boolean shares of the activation value. Thus, in G2, the secret is S and we need to ensure glitch-extended probing security with respect to S. We use the DOM-indep AND gate primitive in the design and register its outputs. Registering the outputs of the compression step allows for its secure composition in the glitch-extended probing model [FGP+18]. To give an intuition behind this claim, the isolated inner-product term is no longer observable with a glitch-extended probe at the gate output—the observation set consists of only an XOR sum of the inner-product term and the refreshed cross-product, which refreshes the entire sum.

Theorem 2. *G2 is glitch-extended probing secure given S as the secret.*

Proof. (*Sketch*) $\mathcal{A}$ can either probe one of the outputs of the `Share Creation` block (Case-I), or the output from one of the tree nodes (Case-II).

- **Case-I**: The outputs of the `Share Creation` block are registered. If $\mathcal{A}$ probes the input to the i^{th} register, the observation set $\mathcal{O}_{[i]} = \{a_{[i]}, r_{[i]}\}$. $a_{[i]}$ is the i^{th} bit of the 15-bit arithmetic share of S, which is independent of $S_{[i]}$. $r_{[i]}$ is the output of the PRNG which is also independent of S. Thus, any arbitrary function of the elements of $\mathcal{O}_{[i]}$ will also produce an output independent of $S_{[i]}$.

- **Case-II**: Figure 15 shows two types of nodes: the generate and propagate units in the stage-1, and the group generate and group propagate units in further stages.

  1. *Generate and propagate nodes (level-0)*: g_i is defined as the tuple consisting of the two outputs from the DOM-indep AND gate. The outputs of a DOM-indep AND gate are independent of its inputs and the computation is secure in the presence of glitches[10]. Thus, the secrets $a_{[i]}$ and $b_{[i]}$, and consequently $S_{[i]}$ are not revealed using glitch-extended probes at any point in the g_i computation. Since every g_i is computed the same way and uses a fresh random share, we prove by induction that every g_i computation is secure in the presence of glitch-extended probes.

     The p_i tuple is equal to the XOR sum of the respective domain terms, i.e., $(p_i^1, p_i^0) = (a_{[i]}^1 \oplus b_{[i]}^1, a_{[i]}^0 \oplus b_{[i]}^0)$. Probing either share $p_{[i]}^0$ or $p_{[i]}^1$ of p_i generates the observation sets $\{a_{[i]}^0, b_{[i]}^0\}$ and $\{a_{[i]}^1, b_{[i]}^1\}$, respectively. Since both sets only contain terms from the same domain, any arbitrary function of these terms will generate an independent output from the unshared secrets $a_{[i]}$ and $b_{[i]}$. G2 registers p_i^0, p_i^1 to restrict the propagation of glitches to the next stage.

  2. *Group generate and group propagate nodes (level-i>0)*: Probes on inputs only reveal one share of either g_i or p_i, which are independent of the secrets as already discussed for level-0.

     For level-1 in Figure 9, the input group generate terms $s_{i:k}, s_{k-1:j}$ and group propagate terms $t_{i:k}, t_{k-1:j}$ are simply the generate g_i and propagate p_i outputs from level-0, with $k = i$ and $j = k - 1$. For instance, if $i = 1$, $s_{1:1} = g_1$, $t_{1:1} = p_1$, $s_{0:0} = g_0$, and $t_{0:0} = p_0$. Using this information in Equation (9), $(s_{1:0}^1, s_{1:0}^0) = (u^1 \oplus g_1^1, u^0 \oplus g_1^0)$. Probing either share of $s_{1:0}$ only reveals the same domain shares of u and g_1. Both shares of u and g_1 are direct registered outputs from DOM-indep AND gate, which implies that they are computed securely and are independent of the unshared inputs. Thus, computation of $s_{1:0}$ is also secure and does not reveal any part of the secret S.

     The group propagate tuple $(t_{1:0}^1, t_{1:0}^0)$ is produced directly as an output of a DOM-indep AND gate with tuples $(t_{1:1}^1, t_{1:1}^0)$ and $(t_{0:0}^1, t_{0:0}^0)$ as inputs. Thus, the

[10]i.e., any arbitrary function of variables in all possible observation sets in the gadget outputs a variable independent of any of the secrets.

computation of $t_{1:0}$ also does not reveal the secret S in the presence of glitches due to the security of the DOM-indep AND gate with registered outputs.

Every level has a distinct number of group generate and group propagate nodes, but processes the outputs from the previous level exactly like level-2. Every level also feeds a fresh and random mask to the DOM-indep AND gates. Also, since the hardware registers the output of DOM gates, chaining them do not lead to any composability flaws. Thus, we prove by induction that all the intermediate computations in the remaining levels do not reveal the original secret S in the glitch-extended probing model.

□

8.3 Boolean-to-arithmetic conversion

The inputs to this gadget `G3` are 1-bit Boolean shares (x^0, x^1) of the activation value x and output is a 15-bit value a such that $a + x^1 = x^0 \oplus x^1$. We provide the *probing security* guarantee that none of the intermediate nets leak the value of the original secret x in the process of generating a.

Theorem 3. *All the intermediate nets in* `G3` *are independent of* x.

Proof. (*Sketch*) The gadget pads the inputs before feeding them to the *Pipelined Golic's B2A* block (see Figure 10). We first prove that the padding is secure and then prove the security of the B2A circuit.

1. *Padding.* The gadget pads both x^0 and x^1 with a 14-bit fresh and uniformly sampled r to produce $y^0 = r||x^0$ and $y^1 = r||x^1$; the operator $||$ denotes concatenation. The LSB $y^i_{[0]} = r_{[0]}||x^i$. Both x^i and $r_{[0]}$ are independent of x and so is the net $r_{[0]}||x^i$. Therefore, the all the intermediate nets in this operation are independent of the secret x.

   For the other bits, $y^i_{[j]} = r_{[j]}$ where $j > 1$ and $i \in \{0, 1\}$. Since $r_{[j]}$ is a fresh and uniformly sampled random bit, all the nets for the rest of the bits are also independent of x.

2. *Pipelined Golic's B2A circuit.* The analysis of padding step proves that all the inputs to this circuit are independent of x, so we exclude the input probes in this analysis. We start our analysis from the LSB of a, i.e., stage-0. $a_{[0]} = y^0_{[0]}$, and since $y^0_{[0]}$ is independent of x, $a_{[0]}$ is independent of x.

   In stage-1, the XOR output is $y^0_{[1]} \oplus y^1_{[0]}$. Since $y^0_{[1]}$ and $y^1_{[0]}$ are the Boolean shares of $y_{[1]}$ and $y_{[0]}$, respectively, their XOR sum will yield another uniform random value because the two bits $y_{[1]}$ and $y_{[0]}$ are independent of each other.[11].

   The output of the multiplexer in stage-1 is $(a_{[0]} \cdot (y^0_{[1]} \oplus y^1_{[0]})) \mid (a'_{[0]} \cdot y^0_{[1]})$. All the variables in this equation are uniformly distributed random numbers. Thus, the probability distribution of the multiplexer output is also independent of x.

   From stage-2 onwards, $y^i_{[j]}$, with $i \in \{(0, 1\}$, and $0 < j < 15$, is a fresh and uniform random number. The other input to the stage is the output $a_{[j]}$ from the previous stage. We proved in the analysis of stage-1 that $a_{[1]}$ is independent of x. Thus, from stage-2 onwards all the intermediates nets are independent of the secret x.

□

[11]We need to be careful here. If the two variables were Boolean shares of the same bit of y, then $\mathcal{A}$ can easily recreate the secret by the shares, even though they're independent of each other.

8.4 Masked Output Layer

Arithmetic to Boolean Conversion. The inputs to this gadget G4 are 15-bit arithmetic shares of the output layer confidence scores z^0 and z^1. Thus, there is no need for random padding like the Boolean-to-arithmetic share conversion. The circuit only consists of the *Pipelined Golic's B2A* block from Figure 10. This block receives arithmetic shares of the output layer confidence scores. Therefore, each bit of the input is independent of the unshared confidence score bits. We have already proved that the intermediate nets in the *Pipelined Golic's B2A* circuit are independent of the unshared inputs in stages 0 and 1. Thus, in the following proof, we directly start from stage-2.

Theorem 4. *All the intermediate nets in G4 are independent of z.*

Proof. We call the inputs to the *Pipelined Golic's B2A* block as z^0 and z^1 in this proof, instead of y^0 and y^1 in Figure 10 because the circuit is the same but in this case the hardware drives arithmetic shares (z^0, z^1) instead of Boolean shares (y^0, y^1).

We first analyse the XOR output of $z^0_{[2]}$ and $z^1_{[1]}$ in stage-2. Since $z^0_{[2]}$ and $z^1_{[1]}$ are Boolean shares of $z_{[1]}$ and $z_{[2]}$, respectively, their XOR sum is independent of both $z_{[1]}$ and $z_{[2]}$.

Next we analyse the XOR output of $z^0_{[1]}$ and $z^0_{[2]}$. Since $z^0_{[1]}, z^0_{[2]}$ are Boolean shares of $z_{[1]}$ and $z_{[2]}$, respectively, their XOR sum is independent of both $z_{[1]}$ and $z_{[2]}$.

Next we analyse the output of the multiplexer in stage-2, which is either $z^0_{[1]} \oplus z^0_{[2]}$ or $z^0_{[2]} \oplus z^1_{[1]}$. We have already proved that the XOR outputs of stage-2 are independent of the secret z and uniformly distributed random numbers. Thus, the multiplexer output is also independent of the secret z.

All the subsequent stages have the same construction. The inputs for every stage i are $z^0_{[i-1]}$, $z^0_{[i]}$, $z^1_{[i-1]}$, and $a_{[i-1]}$. Thus, by induction, the intermediate nets of all the subsequent stages are independent of the unshared input z.

□

Masked Threshold. Section 6.4 describes that the masked threshold block is 14 parallel instantiations of a DOM-indep AND gate with registered outputs and a random sharing of zero. Faust et al. already showed the DOM-indep AND gate with registered outputs to be secure in the glitch-extended probing model. Thus, the masked threshold gadget is also secure in the same model and we skip the security proof for this gadget.

Masked Comparator. The masked comparator gadget G5 takes as input two 15-bit Boolean shares (a^0, a^1) and (b^0, b^1) of the operands a and b to be compared and a 15-bit fresh and random mask r_2. The secrets in this case are the unshared operands a and b, and the gadget needs to ensure that none of the intermediate nets directly depend on either a or b. Figure 12 shows that G5 is a tree structure purely composed of DOM-indep AND gates with registered outputs at each node. Referring to the work by Faust et al. again, we can directly say that G5 is also secure in the glitch-extended probing model.

Masked Multiplexer The inputs to the masked multiplexer gadget G6 are the Boolean shares (a^0, a^1), (b^0, b^1), and (s^0, s^1) of the unshared data and select inputs a, b, and s, respectively. The secrets are the unshared data and select inputs. Thus, the gadget needs to ensure that all the intermediate nets are independent of a, b, and s. As discussed in Section 6.4, the masked multiplexer is a trivial and small gadget with 3 AND gates connected in succession replaced by the masked AND counterparts. Since the masked primitive is secure in the glitch-extended probing model, G6 is also secure.

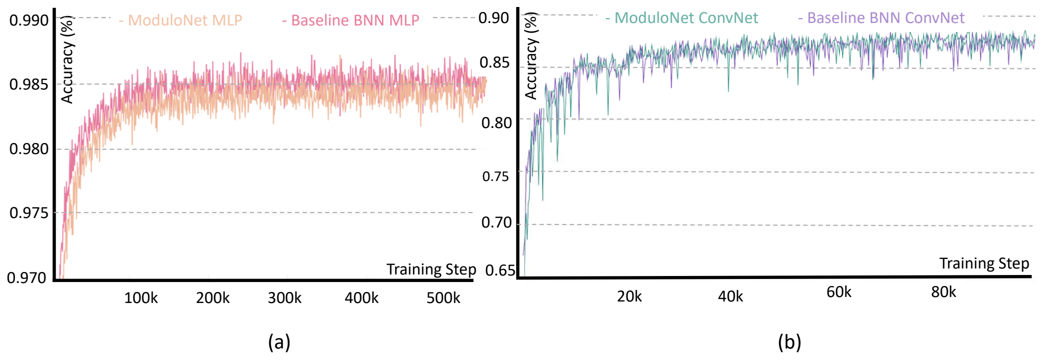


Figure 16: Test accuracy comparison of BNN with baseline vs. ModuloNET for (a) MLP-MNIST, and (b) ConvNet-CIFAR10 datasets. Vertical axis shows the accuracy in % while the horizontal axis displays the steps.

8.5 Proofs for ConvNet Gadgets

The ConvNet uses `G1` to mask the convolutions and DOM-indep AND gate to mask the maxpool operation. We already prove that these gadgets are secure in the presence of glitch-extended probes. The rest of the hardware in masked ConvNet simply reuse the masked activation function and the masked output layer blocks of MLP. We already provide a security proof of the masked activation function and the components of the masked output layer. Therefore, we do not have any additional gadgets in the ConvNet design that require proof.

9 ModuloNET Results and Comparison

In this section, we present the accuracy results of the software implementation with modular arithmetic, the FPGA implementation results of the proposed masked hardware design, and our side-channel evaluations.

9.1 Software Accuracy Results

Evaluation Setup. Our MLP consists of 4 dense layers of 4096 nodes each, while the ConvNet consists of 6 conv layers with number of channels [128, 128, 256, 256, 512, 512], and 3 dense layers with 1024 nodes each, similar to the baseline BNN architecture [CB16]. We utilized square-hinge loss with exponential learning rate decay, minimized using the Adam Optimizer for both networks. We utilized a single c240g5 compute node on CloudLab [DRM$^+$19]—integrated with an NVIDIA Tesla P100 GPU—to run our benchmarks. We also use Weights and Biases (W&B) [Bie20] to log, track, and visualize our experiments.

Accuracy Results. Figures 16 (a) and (b) show accuracy comparison plots with and without modular arithmetic for MLP-MNIST and ConvNet-CIFAR10, respectively. Our ConvNet implementation demonstrates more uncertainty during the early stages of training but becomes comparable when the network stabilizes. This is because the BN effect is more pronounced for deeper architectures which are more difficult to train [IS15]. Thus, the ConvNet experiences additional challenges compared to MLP (6 layers of ConvNet vs 4 layers of MLP, more complex dataset CIFAR10 vs MNIST). Using modular arithmetic has marginal impact on the accuracy—ModuloNET achieves accuracy of 98.77% and 87.83% on MNIST and CIFAR10, worse than the baseline BNNs by 0.27% and 0.77%, respectively.

Table 1: Accuracy Comparison of ModuloNET against BNNs, Full-precision and state-of-the-art networks.

Work	MLP-MNIST	ConvNet-CIFAR10
Full-Precision, State-of-the-Art (With augmentation / other auxiliary features)		
APAC [SNY15]	99.74%	89.67%
EffNet-L2+SAM [FKMN20]	-	99.70%
Full-Precision, Standard Networks (Without bells-and-whistles)		
Maxout+Dropout [GWFM+13]	99.06%	88.32%
Binary Precision of Weights+Activations		
BNNs [CB16]	99.04%	88.60%
ModuloNet	98.77%	87.83%

We also present our results for the CIFAR100 dataset in Figure 21. Such trade offs between accuracy and efficiency are common in quantized ML literature [Guo18].

Table 1 shows accuracy comparison between ModuloNet with BNNs, standard networks without any bells-and-whistles, and the state-of-the-art networks with auxiliary features. Maxout Networks [GWFM+13] proposed a maxout pooling operation alongside the well known dropout regularization to leverage approximate model averaging. The best accuracy without any data-augmentation is reported to be 99.06% and 88.32% on MNIST and CIFAR10, respectively.

In state-of-the-art networks that utilize several auxiliary elements in their networks, Sharpness-aware minimization (SAM) [FKMN20] explores minimization of both loss value and sharpness as optimization objectives. Combined with the network architecture of EfficientNet [TL19], SAM is able to achieve test accuracy of 99.70% on CIFAR10 dataset. APAC [SNY15] proposed a decision-rule for augmented data learning aiming at improving the robustness against intra-class variation during training. The proposed method resulted in accuracy of 99.74% on the MNIST dataset.

Full-precision networks can achieve better performance than binary-precision networks given their better representational capacity. Furthermore, complex network architectures, data augmentation, skip connects, improved loss functions and other auxiliary features are also frequently utilized to improve accuracy. These improvements are tangential to our discussion and can potentially be employed on top of ModuloNet to boost the accuracy.

9.2 Comparison of Masking Overheads

Table 2 presents the area costs of the ModuloNET MLP and ConvNet designs. We compare the area overheads of ModuloNET MLP with the current state-of-the-art[12] that used purely Boolean masking [DCA20a], and also with the baseline unmasked design; there is no prior work on masking a ConvNet. We design the MLP hardware with 3 hidden layers and L=784, M=1024, and N=10 to closely compare with the state-of-the-art design [DCA20a] that had L=784, M=1010, and N=10. All the MLP designs achieve virtually the same[13] latency of approximately 2.9 million cycles. By utilizing modular computations and arithmetic masking, we significantly reduce the LUT and FF overheads from 5.4× and 6.8× in the prior solution [DCA20a] to 3× and 4.5×, respectively.

We emphasize that the reductions in the area of ModuloNET are largely due to the use of arithmetic masking. We quantify this by calculating the cost of masking an N-bit adder using Boolean vs arithmetic masking. From BoMaNet [DCA20a], the approximate area cost of a 15-bit Boolean masked adder[14] is 716 LUTs and 788 FFs, plus it requires 45 fresh

[12]Note we are only comparing hardware that claim first-order side-channel resilience with masking.

[13]The masked components add some additional cycles but the percentage increase is minimal due to the already high latency of the sequentialized design.

[14]BoMaNet used a 20-bit adder but we use 15 bits in our calculations for a fair comparison.

Table 2: Area and Performance Comparison of ModuloNET Designs

MLP Designs	LUT	FF	Latency	PRNGs[4]	(L,M,N)
Baseline-MLP [1]	1833	1125	2913294	0	(784,1024,10)
BoMaNET [2][DCA20a]	9833	7624	2938286	73	(784,1010,10)
ModuloNET-MLP [2]	**5635**	**5009**	**2914134**	**96**	(784,1024,10)
ConvNet Designs	LUT	FF	Latency	PRNGs[4]	(L,P,Q,M,N)[3]
Baseline-ConvNet [1]	3517	1223	160986	0	(256,4,2,1024,10)
ModuloNET-ConvNet [2]	**6583**	**5070**	**161847**	**96**	(256,4,2,1024,10)

[1] unmasked; [2] masked; [3] P and Q denote the kernel and maxpool sizes, respectively;
[4] number of bits/cycle.

masks per cycle. However, in our design, the masking only requires an additional adder (5-10 LUTs) to process the second arithmetic share and $3\times$ lesser randomness —15 masks per cycle. Therefore, we expect a theoretical reduction of $3\times$ in overheads of ModuloNET compared to that of BoMaNet [DCA20a]. However, in practice, we only see a reduction of $1.8\times$ and $1.5\times$ because of additional components like share conversion circuits, and Boolean masked activation function and output layer, and additional PRNGs.

The masking overheads for the ConvNet shows an interesting trend. The number of LUTs in the baseline ConvNet is $1.9\times$ more than that in the baseline MLP design. This is primarily due to the additional special read logic for the convolution and maxpool operations. However, since most of the masked components are reused from the MLP design in implementing the masked ConvNet, the total area cost for both the masked designs become comparable. Thus, when we compare the costlier baseline ConvNet design with the masked ConvNet design, the LUT overhead is only $1.8\times$. In summary, the LUT overhead is smaller because the ratio of control is more in ConvNet due to control logic for convolution and maxpool.

To quantify how our designs scale for more complicated networks, we have also synthesized our masked MLP hardware for hyperparameters of 256, 512, 1k, 2k, and 4k nodes per hidden layer for a larger FPGA [15]. Figure 17 shows the variation. The LUT, and FF costs scales linearly with the increase in this hyperparameter and the latency varies non-linearly because in our sequentialized design, it is roughly equal to $(L+1) \times M + 2 \times M(M+1) + (M+1) \times N$ [16]. Note that having more/deeper hidden layers do not affect the area-cost but increases the latency linearly.

9.3 Side-Channel Results

Measurement Setup and Network Configuration. We use Xilinx ISE 14.7 for synthesis and bitstream generation and utilize the *DONT_TOUCH* attribute to ensure that the design is synthesized the way it is coded, without optimizations, for security. We use the Sakura-G board as our side-channel analysis platform that hosts a Xilinx Spartan-6 FPGA (Device: XC6SLX75, Package: CSG484, Speed:-2) to execute the hardware implementation to be evaluated and has a designated SMA port that provides the power drop across a shunt resistor of 1Ω on the main supply line. We use Picoscope 3206D as the oscilloscope to acquire the voltage drop. We set a very low design frequency of 1.5 MHz to reduce information loss due to aliasing between clock cycles. We set the oscilloscope sampling frequency to 125 MHz i.e., the setup captures 83 points per clock cycle. This is a typical setup used for leakage evaluation in prior works on masking [RSM20].

The low operating frequency increases the execution time and slows down the acquisition

[15] Note that our designs LUT and FF are moderate hence they fit into relatively small FPGAs. But even with a binarized network, the on-chip BRAM for storing the weights becomes the bottleneck as the designs scale.

[16] This is true even for the unprotected design.

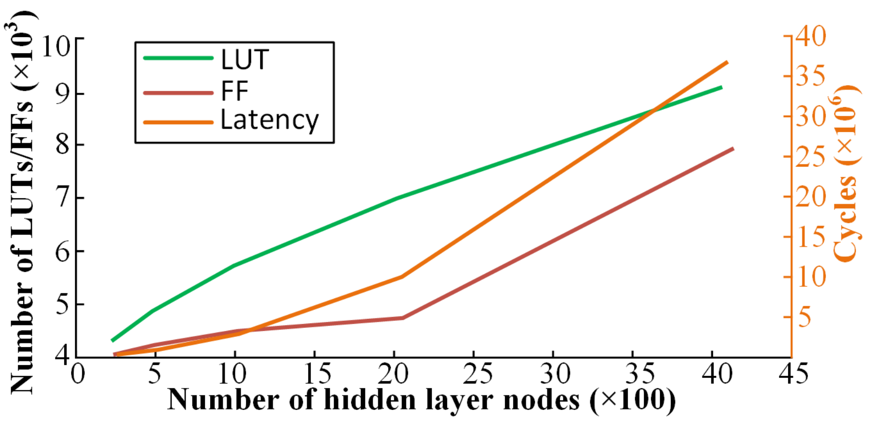


Figure 17: The variation of the number of LUTs, FFs and inference latency with the number of increasing nodes in the hidden layer for the MLP hardware design.

process. The network with 784 input nodes, 3 hidden layers with 1024 nodes and 10 output layer nodes requires roughly 785×1024+1025×3×1024+1025×10=2.9M clock cycles or 1.9 seconds to finish a single inference. Acquiring and evaluating traces at the order of millions for such a network configuration with our high-precision sampling is impractical[17]. Thus, we set our parameterized hardware to 64 nodes in the hidden layers to accelerate the process. Due to the repetitive nature of neural network computations and the sequentialized hardware that computes one node at a time, we argue this configuration to be the canonical example of larger networks with more nodes[18].

Leakage Evaluation of the Masked Design. We adopt the widely-used Test Vector Leakage Assessment (TVLA) methodology to perform the masked design's leakage evaluation [GGJR11]. We do not claim that TVLA is the best methodology for all possible cases [Sta18] but that it is a common technique used in TCHES works to evaluate the security of masking schemes [DAN+18, SEL21, SBM21]. TVLA uses the Welch's t-test to detect the presence of side-channel leakage. Welch's t-test is used in statistics to test the hypothesis that two populations have similar means. The test computes a t-score that is given by the following equation:

$$t = \frac{\mu_1 - \mu_0}{\sqrt{\frac{s_0^2}{n_0^2} + \frac{s_1^2}{n_1^2}}},$$

where s_0, s_1, n_0, n_1 are the variances and the sizes of datasets, respectively. A high t-score results in the rejection of the *null-hypothesis* that states that the two populations are drawn from the same distribution. A t-score crossing the threshold of ±4.5 implies rejection of the null-hypothesis with a confidence of 99.99%, and is the accepted threshold to experimentally detect the presence of side-channel leakage.

We choose the univariate non-specific fixed vs. random t-tests because it is independent of the underlying DUT implementation. In this test, the setup captures two sets of power traces: one in which the input is constant for all executions and one in which it varies per execution. The setup then computes the t-scores over these two sets. A high t-score implies that there is a side-channel leakage in the implementation because the power trace

[17]We lowered frequency for sound leakage evaluation, which increases test time—this is a standard SCA evaluation practice [DAN+18, OSPG18]. Real-world ML accelerators operating at higher frequencies have been attacked [BBJP19, MBC21].

[18]This is, for e.g., practically equivalent to designing a fully masked round-serial AES engine and running it for 10 rounds vs. 12 rounds.

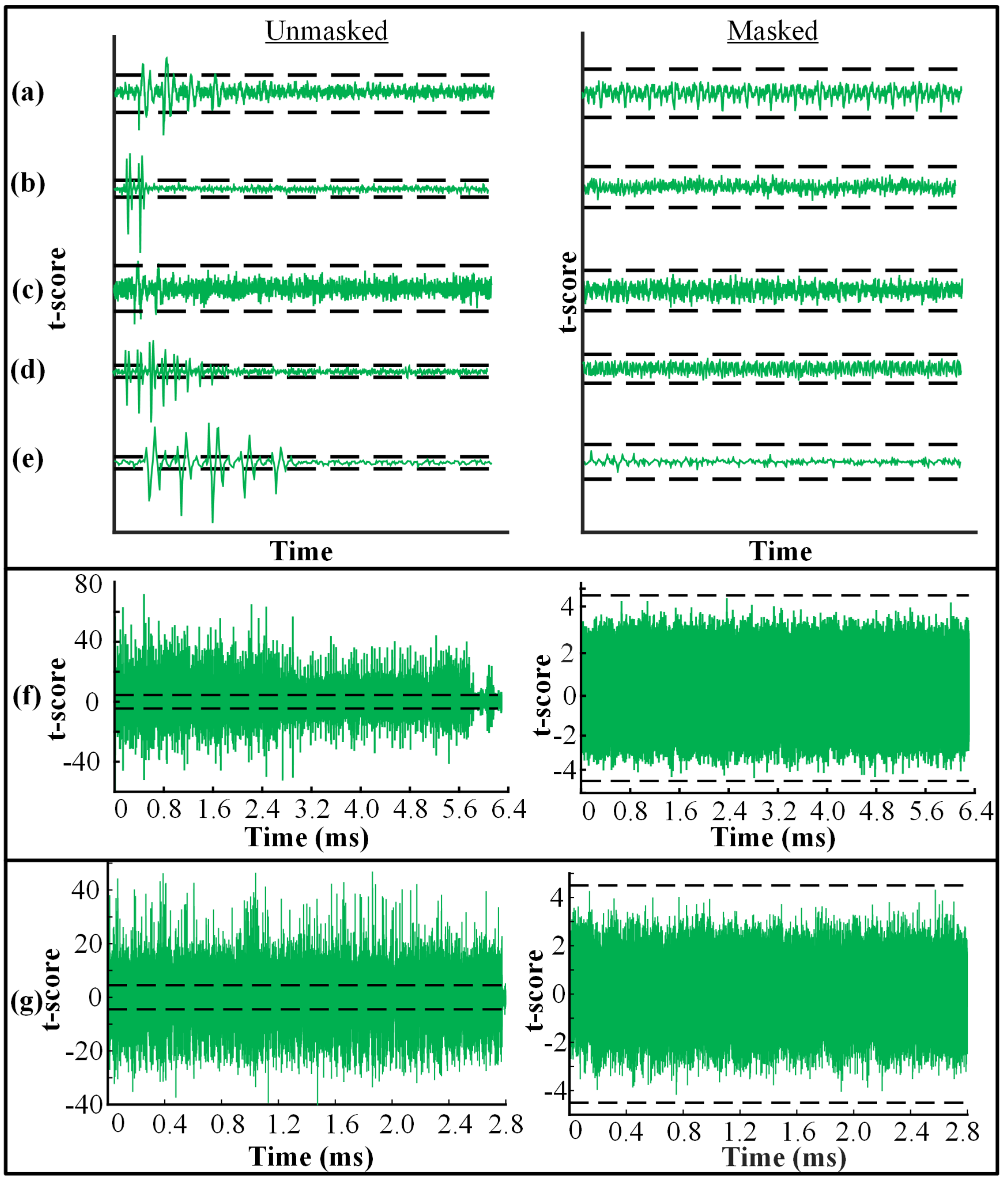


Figure 18: All the plots show TVLA results with PRNGs disabled on the left and PRNGs enabled on the right. The dotted lines signify the TVLA threshold of ±4.5 which statistically implies a 99.99% confidence. Plots (a)-(e) show results for the individual components of the neural network viz. masked activation function, Boolean-to arithmetic converter, arithmetic-to Boolean converter, masked comparator, and masked multiplexer in the same order. The plots (a)-(e) are shown on a common time axis for visual aid; the execution times are different across every component. Plot (f) shows the results for the hidden and output layer computations. We only present the t-scores for hidden and output layer computations because during input layer computations we cannot evaluate a potential leakage directly due to input correlations: the hardware loads and masks the input pixels on-the-fly throughout the processing of the input layer. Therefore, we create another design which buffers all the masked input pixels and the corresponding masks instead of generating them on-the-fly. The hardware directly reads the input pixels only once during the creation of the arithmetic shares. Figure 18 (g) shows the results for this experiment. In all the plots we observe t-scores higher than the threshold for the unmasked design and t-scores within the threshold for the masked equivalents which demonstrates a significant decrease in the side-channel leakage.

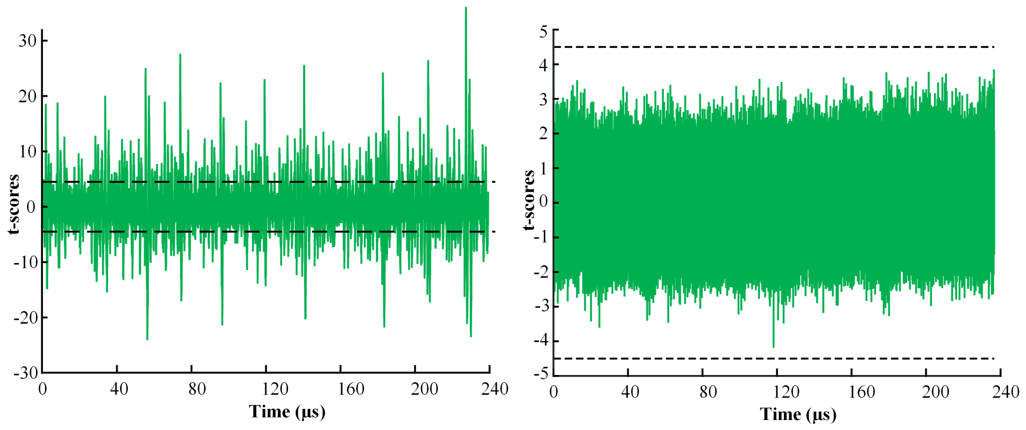


Figure 19: TVLA results of the masked MLP hardware with 10M traces (right). This is evaluated for a small window randomly selected from the overall inference window. The results show that even with an increased number of measurements there is no first-order leakage . The figure on the left is the plot with PRNGs switched off for the same window.

corresponding to a specific input (or fixed dataset) is distinguishable from the general population of power traces (or random dataset). We conduct TVLA for two setups. First, we perform TVLA on the design with disabled PRNGs that has all the fresh masks driven to 0, which is equivalent to an unmasked design. Second, we conduct TVLA on the (equivalently) masked design with enabled PRNGs. Our setup acquires 2M traces, 1M traces each for the fixed and random datasets. Note that this is a typical amount of measurements for designs with long execution times; some recent papers even evaluated using fewer measurements [Sug18, OSPG18].

Figure 18 shows the TVLA results of our experiments. We first evaluate the security of the individual components as a standalone unit. Plots (a)-(e) show the TVLA results for the individual components (respectively for masked activation function, Boolean-to-arithmetic converter, arithmetic-to-Boolean converter, masked comparator, and masked multiplexer), plot (f) shows it for the full design except the input layer, and plot (g) shows this for the input layer only. The results confirm our security claims as the unmasked versions show statistically significant leakages and the masked designs do not.

To further validate the security of our design, we conducted an evaluation of the masked MLP hardware with 10M traces for a small window randomly selected from the overall inference window. Figure 19 shows the the result of this experiment. The t-scores still remain within the threshold of ±4.5 demonstrating the empirical security of our masked design. Note that these experiments are carried out in a low-noise side-channel evaluation platform. Thus, in an actual accelerator the number of traces to break the masking scheme will be significantly higher that 10M due to more noise.

Figure 20 presents our side-channel evaluation results for the binarized ConvNet design. We only capture time samples during convolutional and maxpool operations in this evaluation. The power trace clearly shows the high activity convolution layer computation followed by a low activity OR-based maxpool operation and the start of FC layer activity towards the end. The design with PRNGs disabled clearly leaks with t-scores crossing the ±4.5 threshold throughout the inference. By contrast, the t-scores never cross the threshold when the PRNGs are enabled effectively masking the design. Thus, we demonstrate the first-order side-channel security of the masked binarized ConvNet.

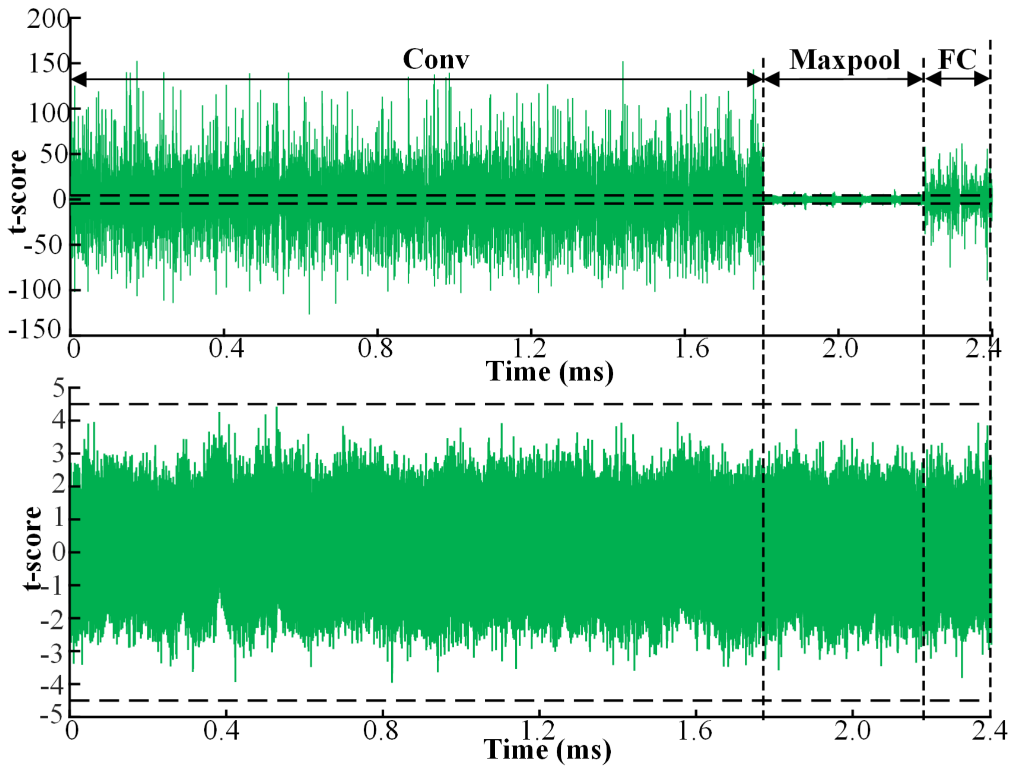


Figure 20: TVLA results of the masked ConvNet hardware with 1M traces. When the PRNGs are off (i.e., the design is effectively unprotected), the design has significant side-channel leakage as expected. However, when the PRNG is on (i.e., the design is masked), there is no first-order leakage, i.e., empirical results match our theoretical proofs of security.

10 Discussions

In this section, we discuss some potential refinements in masking, limitations of the leakage evaluation and the proposed learning scheme, and orthogonal works.

Empirical vs. Provably-Secure Masking. We aim at both theoretical and empirical security in our design to make model extraction as hard as theoretical attacks. We do realize the importance of security notions and proofs. Various security models have been proposed to formally evaluate a masked gadget in an ideal setting [ISW03], or real setting [PR13]. Some newer models have also been proposed specifically to model glitches and transitions on hardware [FGP+18, CS21]. Others construct notions that help to prove secure composition of the masked *gadgets* like NI, SNI, and PINI [BBD+16, CS20]. We provide a theoretical security analysis in the robust-probing model for this work. Future works can certainly explore further on optimizing the gadgets while still maintaining composability guarantees. Alternatively, we can use automated tools to this end [KSM20], but as noted in a recently published work [SBM21], no tool can verify the security of a full cipher and thus, we need to rely on the empirical analysis along with the theoretical guarantees that the tools may provide.

TVLA vs Other Methods. In addition to the TVLA, there are several alternatives to evaluate the side-channel leakage in a proposed implementation [KJJ99, BCO04, CRR03, GBTP08, HGD+11]. These typically need the implementation/algorithm knowledge to make hypothesis and construct a power model or require re-configuring secret information and building a power profile of the target device. We choose TVLA, which does not have these limitations. However, we are aware of the TVLA's drawbacks due to its simple moment-based analysis that can lead to false-negative and -positives [Sta18]. Welch's t-test can be complemented with the χ^2 test to capture information lying in multiple moments [MRSS18]. But the χ^2 test is mostly used for validating higher-order masking schemes or threshold implementations, hence, we consider it out of scope for this work.

Masking Enhancements and Limitations. Our proposal to use modular arithmetic for neural networks has already caused significant reductions over the state-of-the-art, but there is certainly scope for improvement. Advanced schemes such as GLM, UMA, etc. can be used to further reduce the latency and randomness of the masked gadgets [GIB18, GM18, RSM20]. Optimized share conversion circuits can also be incorporated to reduce the area costs [MTMM07b, Deb12]. The current design uses Boolean masking in the output layer, but it would be interesting to explore ways to perform operations like comparisons only with arithmetic shares. A recent work on secure multiparty computation explores something similar [MRVW21].

Recent work demonstrated how first-order secure implementations on an FPGA can become vulnerable under high temperature, high voltage, and high clock frequency due to coupling effects [DEM18]. Levi et al. [LBS19] further demonstrated how the couplings can be externally amplified to break the assumptions of provable security. Morover, most of the proposed schemes suffer from local or global compositional flaws [MMSS19] when extended to higher-order. Higher-order masking with related challenges and attacks that manipulate the setup is out of scope, i.e., we exclusively focus on first-order masking as in prior recent works [RSM20, SBM21]. More maturity is needed on these aspects for cryptography use cases before investigating and transitioning them to the ML world. To that end, we expect main challenges to be building efficient and secure maxpool, ReLU, and output layer comparator masked gadgets by borrowing/augmenting the lessons learned from extending masked gadgets from first-order to higher-order [MMSS19].

Drawbacks of Proposed Layer Architecture. Our approach relies on lossless modulo folding. In principle, the modulo operation entails the loss of information but unfolding and regenerating the original samples is possible under the condition that the modulo is performed using a large enough K (Using Equation (5)) such that modulo does not scramble the samples. This has two major disadvantages:

1. The selection of K is dependent on the input data distribution. While K can be selected using the training/validation data, its applicability to the test split is dependent on how similar the test split is to the train/validation split.

2. Smaller K translates to lower probability $Pr(O_n)$ which leads to degradation in accuracy since the network starts to lose its representation capacity. In our experiments, we utilized modulo to effectively shift the representation rage of data from $\left(-\frac{K}{2}, \frac{K}{2}\right)$ to $(0, K)$. This allows us to circumvent the sign leakage problem while maintaining accuracy. Future works could explore contracting the representation range by reducing K and encoding the 'number of wraparounds', $\lfloor \frac{x}{K} \rfloor$, within the network, allowing compressed propagation of information.

Analogy with Fully-Homomorphic Encryption Schemes. By definition, all homomorphic encryption (HE) schemes work with modular arithmetic; our novelty is to perform such computations directly on the plaintext for the 'full inference'. Bourse et al. [BMMP18] published a work on using HE for discretized neural networks (DiNN)–networks with discrete weights and sign-function as the activation function. Our advantage over prior works using HE is to evaluate the full function (including output layer which is excluded in DiNN, and without input discretization), supporting a more diverse set of operations e.g., batch normalization and output-max operation, and supporting more diverse layers such as convolutional and maxpool.

Scalability with Other Datasets and Network Sizes. Our software and hardware techniques are not fundamentally limited by the complexity of the datasets being used, or the network size. In the context of ML security/privacy research, MNIST and CIFAR10 are the two de facto standards in major security and ML conferences like ICML, USENIX, CCS, and CRYPTO to demonstrate the proof of concept [DGBL+16, JVC18, MR18, RSC+19, CJM20]. Our masked NNs do support these two datasets. Any new dataset will naturally have a Gaussian distribution and require a re-tuning of the modulus value via the techniques described in Section 4 . The modulus value decides the size of the field and thus, the randomness requirements to mask the weighted summations or convolutions.

Increasing the number of FC or convolutional layers, or the number of nodes per layer does not affect the masking-related overheads in our hardware architecture because the design reuses the masked components. The area will certainly increase with the increasing network size as evidenced by our experiments because of increased storage elements and additional counter bits in the controller to account for the additional nodes or layers. Interestingly, as the networks become more complicated, the bottleneck of the design seems to move towards on-chip memory storage (rather than datapath size). Therefore, if there is insufficient on-chip memory to store all network parameters unlike our case, the challenge is likely to be on the data movement, e.g., moving data from external memory to FPGA fabric with high-throughput DDRs. Our hardware designs can be parallelized/replicated as black-box instances to match the throughput of the data movement.

Side-Channel Attacks versus Theoretical Model Extraction. Orthogonal to the side-channel attacks, there has also been work on model stealing by theoretically analyzing the predictions of the model over chosen inputs [JCB+20, CJM20, TZJ+16]. The attacks typically perform black-box queries to the model to create a surrogate model with high fidelity and/or high accuracy. The state-of-the-art theoretical attack needs at least $2^{21.5}$ queries to extract the parameters from a three-layer neural network with 784, 128, and 10 nodes in each layer and do not yet extend to more complicated networks. In contrast, the side-channel attacks on neural networks can extract the parameters with only 40k queries for a much larger 4-layer neural network [DCA20b]. Our analysis validates the side-channel security with about 2^{20} queries/side-channel tests, raising the difficulty of breaking to the level of mathematical cryptanalysis.

Protecting the Embedded ML Applications A lot of neural network development is also done at the software level and deployed on microcontroller-based targets, which is are prime targets for side-channel attacks. A potential solution is to employ software masking for C-based implementations like the AES-based works [MOPT12, ABP12, BRB+11, BDM+20, SSB+21]. For higher-level language(e.g., Python), one can first convert it to C (e.g., via Cython[19]) and then apply such defenses. Additionally, one must be careful about the unintended interactions in the microarchitecture leaking the secrets as discussed in ROSITA [SSB+21], FENL [GMPP20], and other research works.

[19]https://cython.org/

11 Conclusion and Future Work

ML algorithms provide a new avenue for side-channel analysis research. While the majority of the existing work focuses on potential attacks, the defenses are largely overlooked. This paper reveals that the opportunities in building efficient masking schemes do not *only* lay in tuning the application of masking but further in tweaking the target algorithm itself—unlike in cryptographic standards, ML has this flexibility. The results quantify that such an algorithm/hardware co-design with side-channel security in mind can significantly reduce the area cost with a marginal effect on accuracy.

This paper aims to build efficient masking schemes and to both *empirically* validate their security with practical side-channel tests and *theoretically* validate with proofs. Our defense is the most efficient one known to date among comparable solutions. However, both the proposed attacks and the defenses can be extended, e.g., to higher-order masking, provably-secure masking, and attacks that can make such proofs obsolete. Even today, extending masking and related attacks are subjects undergoing intense study after decades of evaluation of such attacks and defenses on the well-established cryptographic standards. There are many ways to construct neural networks, just like there are many ways to construct cryptosystems. Our goal in this paper is not to mask all possible network configurations but to investigate secure masking optimizations in-depth for one particular network style. As we demonstrated in this work, doing so for a particular configuration is quite challenging and involved. Nevertheless, we also present our results on masking ConvNets and its associated challenges. Future works can explore the masking of other types of networks like the recurrent NNs, networks with optimizations such as skip-connects, or even other types of classifiers such as decision trees, support vector machines, etc.

12 Acknowledgements

This project is supported in parts by NSF under Grants No. 1943245 and SRC GRC Task 2908.001. This work was also partially funded by Lahore University of Management Sciences (LUMS) through Grant-ID FIF-569. The authors are grateful for this support. We thank the CHES reviewers and our shepherd Dr. Lauren De Meyer for their valuable feedback that greatly improved the quality of our manuscript. We also thank Dr. Vikram Suresh, Dr. Sohrab Aftabjahani, Dr. Avinash Varna, Dr. Raghavan Kumar, and Dr. Amitabh Das for their expert guidance throughout the project. We are grateful to the Cloudlab [DRM+19] team for compute support and to the W&B [Bie20] team for logging and visualisation support.

References

[ABP12] Giovanni Agosta, Alessandro Barenghi, and Gerardo Pelosi. A code morphing methodology to automate power analysis countermeasures. In *Proceedings of the 49th Annual Design Automation Conference*, pages 77–82, 2012.

[ABP+18] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic keccak: SCA security and low latency in HW. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):269–290, 2018.

[AG01] Mehdi-Laurent Akkar and Christophe Giraud. An implementation of DES and AES, secure against some attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 309–318. Springer, Heidelberg, May 2001.

[AGM$^+$09] Monjur Alam, Santosh Ghosh, MJ Mohan, Debdeep Mukhopadhyay, Dipanwita Roy Chowdhury, and IS Gupta. Effect of glitches against masked AES S-Box implementation and countermeasure. *IET Information Security*, 3(1):34–44, 2009.

[BBD$^+$16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.

[BBH$^+$19] Shivam Bhasin, Jakub Breier, Xiaolu Hou, Dirmanto Jap, Romain Poussier, and Siang Meng Sim. SITM: See-in-the-middle side-channel assisted middle round differential cryptanalysis on spn block ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):95–122, Nov. 2019.

[BBJP19] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSI NN: reverse engineering of neural network architectures through electromagnetic side channel. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 515–532. USENIX Association, 2019.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, Heidelberg, August 2004.

[BDM$^+$20] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In *Eurocrypt 2020-39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 12107, pages 311–341. Springer, 2020.

[Bie20] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[BIL$^+$15] Carlo Baldassi, A. Ingrosso, Carlo Lucibello, Luca Saglietti, and R. Zecchina. Subdominant dense clusters allow for simple learning and high computational performance in neural networks with discrete synapses. *Physical review letters*, 115 12:128101, 2015.

[BJH$^+$21] Jakub Breier, Dirmanto Jap, Xiaolu Hou, Shivam Bhasin, and Yang Liu. Sniff: Reverse engineering of neural networks with fault attacks. *IEEE Transactions on Reliability*, 2021.

[BLC13] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512. Springer, 2018.

[BRB$^+$11] Ali Galip Bayrak, Francesco Regazzoni, Philip Brisk, François-Xavier Standaert, and Paolo Ienne. A first step towards automatic application of power analysis countermeasures. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 230–235. IEEE, 2011.

[CB16] Matthieu Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *ArXiv*, abs/1602.02830, 2016.

[CBD15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 3123–3131, Cambridge, MA, USA, 2015. MIT Press.

[CEvMS15] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Differential power analysis of a McEliece cryptosystem. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 538–556. Springer, Heidelberg, June 2015.

[CJM20] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 189–218. Springer, 2020.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Heidelberg, August 2003.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):136–158, Feb. 2021.

[DAN$^+$18] Lauren De Meyer, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. M&M: Masks and macs against physical attacks. *IACR TCHES*, 2019(1):25–50, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7333.

[DCA20a] Anuj Dubey, Rosario Cammarota, and Aydin Aysu. Bomanet: Boolean masking of an entire neural network. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020*, pages 51:1–51:9. IEEE, 2020.

[DCA20b] Anuj Dubey, Rosario Cammarota, and Aydin Aysu. Maskednet: The first hardware inference engine aiming power side-channel protection. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020*, pages 197–208. IEEE, 2020.

[DDS$^+$09] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[Deb12] Blandine Debraize. Efficient and provably secure methods for switching from arithmetic to Boolean masking. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 107–121. Springer, Heidelberg, September 2012.

[DEM18] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware masking, revisited. *IACR TCHES*, 2018(2):123–148, 2018. https://tches.iacr.org/index.php/TCHES/article/view/877.

[DGBL$^+$16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 201–210. JMLR.org, 2016.

[DRB$^+$16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 194–212. Springer, Heidelberg, August 2016.

[DRB18] Lauren De Meyer, Oscar Reparaz, and Begül Bilgin. Multiplicative masking for AES in hardware. *IACR TCHES*, 2018(3):431–468, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7282.

[DRM$^+$19] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[FBR$^+$21] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked accelerators and instruction set extensions for post-quantum cryptography. *IACR Cryptol. ePrint Arch.*, 2021:479, 2021.

[FGP$^+$18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[FKMN20] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[FMI83] Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (5):826–834, 1983.

[FZS$^+$19] Cheng Fu, Shilin Zhu, Hao Su, Ching-En Lee, and Jishen Zhao. Towards fast and energy-efficient binarized neural network inference on fpga. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, page 306, New York, NY, USA, 2019. Association for Computing Machinery.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 426–442. Springer, Heidelberg, August 2008.

[GGJR11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. 2011. http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf.

[GIB18] Hannes Gross, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR TCHES*, 2018(2):1–21, 2018. https://tches.iacr.org/index.php/TCHES/article/view/871.

[GM18] Hannes Groß and Stefan Mangard. A unified masking approach. *Journal of Cryptographic Engineering*, 8(2):109–124, June 2018.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016.

[GMPP20] Si Gao, Ben Marshall, Dan Page, and Thinh Hung Pham. FENL: an ISE to mitigate analogue micro-architectural leakage. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):73–98, 2020.

[Gol07] Jovan Dj Golic. Techniques for Random Masking in Hardware. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(2):291–300, 2007.

[Gou01] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, Heidelberg, May 2001.

[Guo18] Yunhui Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018.

[GWFM+13] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International conference on machine learning*, pages 1319–1327. PMLR, 2013.

[HCS+17] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[HGD+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, December 2011.

[HZS18] Weizhe Hua, Zhiru Zhang, and G. Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 4:1–4:6. ACM, 2018.

[Ins20] Deloitte Insights. Bringing AI to the device: Edge AI chips come into their own, 2020. https://www2.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/2020/ai-chips.html.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[ISU18] Vincent Immler, Robert Specht, and Florian Unterstein. Your rails cannot hide from localized EM: how dual-rail logic fails on FPGAs - extended version. *Journal of Cryptographic Engineering*, 8(2):125–139, June 2018.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

[JCB$^+$20] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[JSMA19] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N. Asokan. PRADA: protecting against DNN model stealing attacks. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 512–527. IEEE, 2019.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.

[JYI$^+$20] Dirmanto Jap, Ville Yli-Mäyry, Akira Ito, Rei Ueno, Shivam Bhasin, and Naofumi Homma. Practical side-channel based model extraction attack on tree-based machine learning algorithm. In Jianying Zhou, Mauro Conti, Chuadhry Mujeeb Ahmed, Man Ho Au, Lejla Batina, Zhou Li, Jingqiang Lin, Eleonora Losiouk, Bo Luo, Suryadipta Majumdar, Weizhi Meng, Martín Ochoa, Stjepan Picek, Georgios Portokalidis, Cong Wang, and Kehuan Zhang, editors, *Applied Cryptography and Network Security Workshops - ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19-22, 2020, Proceedings*, volume 12418 of *Lecture Notes in Computer Science*, pages 93–105. Springer, 2020.

[KCS$^+$20] Phil C Knag, Gregory K Chen, H Ekin Sumbul, Raghavan Kumar, Mark A Anders, Himanshu Kaul, Steven K Hsu, Amit Agarwal, Monodeep Kar, Seongjong Kim, et al. A 617 tops/w all digital binary neural network accelerator in 10nm finfet cmos. In *2020 IEEE Symposium on VLSI Circuits*, pages 1–2. IEEE, 2020.

[KGB$^+$18] Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. Differential power analysis of XMSS and SPHINCS. In Junfeng Fan and Benedikt Gierlichs, editors, *COSADE 2018*, volume 10815 of *LNCS*, pages 168–188. Springer, Heidelberg, April 2018.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

[KS73] Peter M. Kogge and Harold S. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Computers*, 22(8):786–793, 1973.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.

[LBS19] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. Reducing a masked implementation's effective security order with setup manipulations. *IACR TCHES*, 2019(2):293–317, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7393.

[LHBB99] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In David A. Forsyth, Joseph L. Mundy, Vito Di Gesù, and Roberto Cipolla, editors, *Shape, Contour and Grouping in Computer Vision*, volume 1681 of *Lecture Notes in Computer Science*, page 319. Springer, 1999.

[Lin76] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.

[LWYY20] Xiaoning Liu, Bang Wu, Xingliang Yuan, and Xun Yi. Leia: A lightweight cryptographic neural network inference system at the edge. Cryptology ePrint Archive, Report 2020/463, 2020. https://eprint.iacr.org/2020/463.

[LZP17] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 345–353. Curran Associates, Inc., 2017.

[Mag20] Analytics India Magazine. Why deep learning is a costly affair, 2020. https://analyticsindiamag.com/deep-learning-costs-cloud-compute/.

[MBC21] Saurav Maji, Utsav Banerjee, and Anantha P. Chandrakasan. Leaky nets: Recovering embedded neural network models and inputs through simple power and timing side-channels - attacks and defenses. *CoRR*, abs/2103.14739, 2021.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited. *IACR TCHES*, 2019(2):256–292, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7392.

[MOPT12] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler assisted masking. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 58–75. Springer, 2012.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Josyula R. Rao and Berk Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 157–171. Springer, Heidelberg, August / September 2005.

[MR18] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018.

[MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the χ^2-test. *IACR TCHES*, 2018(1):209–237, 2018. https://tches.iacr.org/index.php/TCHES/article/view/838.

[MRVW21] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. Rabbit: Efficient comparison for secure multi-party computation. *IACR Cryptol. ePrint Arch.*, 2021:119, 2021.

[MTMM07a] Robert McEvoy, Michael Tunstall, Colin C Murphy, and William P Marnane. Differential power analysis of hmac based on sha-2, and countermeasures. In *International Workshop on Information Security Applications*, pages 317–332. Springer, 2007.

[MTMM07b] Robert P. McEvoy, Michael Tunstall, Colin C. Murphy, and William P. Marnane. Differential power analysis of HMAC based on sha-2, and countermeasures. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*, volume 4867 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2007.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS 06*, volume 4307 of *LNCS*, pages 529–545. Springer, Heidelberg, December 2006.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure masked Ring-LWE implementations. *IACR TCHES*, 2018(1):142–174, 2018. https://tches.iacr.org/index.php/TCHES/article/view/836.

[PMG+17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Heidelberg, May 2013.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[RRd+16] Oscar Reparaz, Sujoy Sinha Roy, Ruan de Clercq, Frederik Vercauteren, and Ingrid Verbauwhede. Masking ring-LWE. *Journal of Cryptographic Engineering*, 6(2):139–153, June 2016.

[RSC+19] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: xnor-based oblivious deep neural network inference. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 1501–1518. USENIX Association, 2019.

[RSM20] Aein Rezaei Shahmirzadi and Amir Moradi. Re-consolidating first-order masking schemes: Nullifying fresh randomness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):305–342, Dec. 2020.

[SBM21] Aein Rezaei Shahmirzadi, Dusan Bozilov, and Amir Moradi. New first-order secure AES performance records. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):304–327, 2021.

[Sec20] Security.org. The Best Indoor Cameras for Artificial Intelligence, 2020. https://www.security.org/security-cameras/best/artificial-intelligence.

[SEL21] Okan Seker, Thomas Eisenbarth, and Maciej Liskiewicz. A white-box masking scheme resisting computational and algebraic attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):61–105, 2021.

[SGMT18] Falk Schellenberg, Dennis RE Gnad, Amir Moradi, and Mehdi B Tahoori. An inside job: Remote power analysis attacks on fpgas. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1111–1116. IEEE, 2018.

[SNY15] Ikuro Sato, Hiroki Nishimura, and Kensuke Yokoi. Apac: Augmented pattern classification with neural networks. *arXiv preprint arXiv:1505.03229*, 2015.

[SP06] Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 208–225. Springer, Heidelberg, February 2006.

[SSB+21] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. Rosita: Towards automatic elimination of power-analysis leakage in ciphers. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[Sta18] François-Xavier Standaert. How (not) to use welch's t-test in side-channel security evaluations. In Begül Bilgin and Jean-Bernard Fischer, editors, *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers*, volume 11389 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2018.

[Sug18] Takeshi Sugawara. 3-share threshold implementation of AES s-box without fresh randomness. *IACR TCHES*, 2019(1):123–145, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7336.

[Tec20] Techcrunch. Apple buys edge-based ai startup xnor.ai for a reported $200m, 2020. https://techcrunch.com/2020/01/15/apple-buys-edge-based-ai-startup-xnor-ai-for-a-reported-200m/.

[TG20] Shahin Tajik and Fatemeh Ganji. Artificial neural networks and fault injection attacks. *arXiv preprint arXiv:2008.07072*, 2020.

[TKL04] Elena Trichina, Tymur Korkishko, and Kyung Hee Lee. Small size, low power, side channel-immune aes coprocessor: design and synthesis results. In *International Conference on Advanced Encryption Standard*, pages 113–127. Springer, 2004.

[TL19] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[TSSL20] Go Takatoi, Takeshi Sugawara, Kazuo Sakiyama, and Yang Li. Simple electromagnetic analysis against activation functions of deep neural networks. In Jianying Zhou, Mauro Conti, Chuadhry Mujeeb Ahmed, Man Ho Au, Lejla Batina, Zhou Li, Jingqiang Lin, Eleonora Losiouk, Bo Luo, Suryadipta Majumdar, Weizhi Meng, Martín Ochoa, Stjepan Picek, Georgios Portokalidis, Cong Wang, and Kehuan Zhang, editors, *Applied Cryptography and Network Security Workshops - ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19-22, 2020, Proceedings*, volume 12418 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2020.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.

[UFG+17] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74, 2017.

[WLL+18] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 393–406, 2018.

[YKO+20] Kota Yoshida, Takaya Kubota, Shunsuke Okura, Mitsuru Shiozaki, and Takeshi Fujino. Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[YMY+20] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. DeepEM: Deep neural networks model recovery through EM side-channel information leakage. In *2020 IEEE International Symposium on Hardware*

Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020, pages 209–218. IEEE, 2020.

[YN17] H. Yonekawa and H. Nakahara. On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an fpga. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 98–105, 2017.

[ZCL+19] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.

[ZS18] Mark Zhao and G Edward Suh. FPGA-based remote power side-channel attacks. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 229–244. IEEE, 2018.

[ZWN+16] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

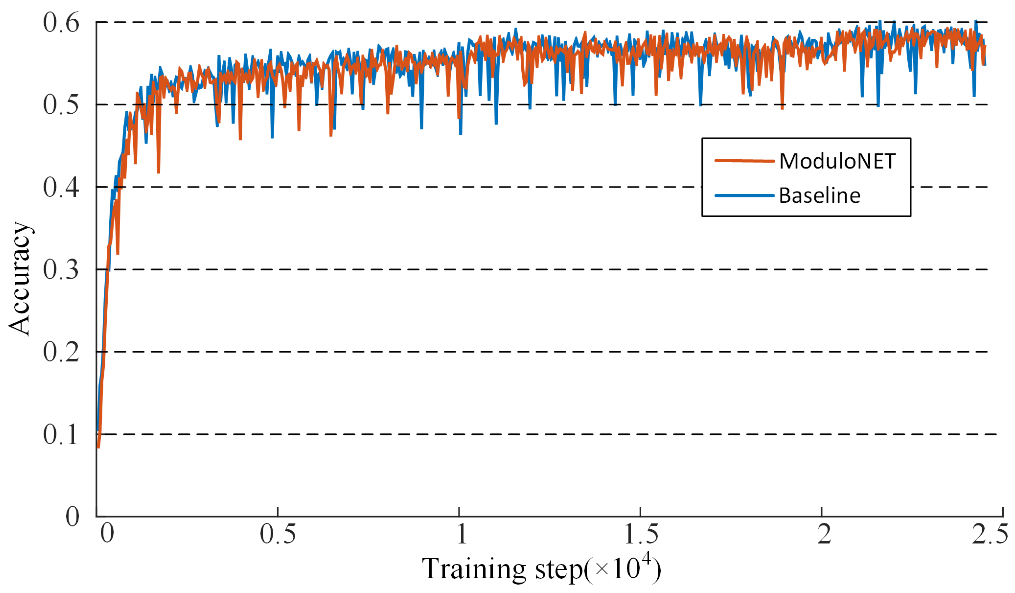


Figure 21: Test accuracy comparison of ConvNet with baseline vs. ModuloNET for the CIFAR100 dataset. The top-1 accuracy for ModuloNET only drop by 0.88% compared to that of the baseline.