A Full-Stack Approach for Side-Channel Secure ML Hardware

Anuj Dubey
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27606

Aydin Aysu
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27606

***Abstract*—Machine learning (ML) has recently emerged as an application with confidentiality needs. A trained ML model is indeed a high-value intellectual property (IP), making it a lucrative target for notorious side-channel attacks. Recent works have already shown the possibility of reverse engineering the model internals by exploiting the side channels like timing and power consumption. But the defenses are largely unexplored. Preventing ML IP theft is highly relevant given that the demand for ML will only increase in the coming years.**

Securing ML hardware against side-channel attacks requires analyzing the vulnerabilities in the current ML applications and developing full-stack countermeasures from the ground up, covering cryptographic proofs, circuit design, firmware support, architecture/microarchitecture integration, compiler extensions, software design, and physical testing. There is a *need to work on all abstraction levels* because focusing on just one or few level(s) cannot provide a complete solution to this nascent problem.

Our research achieves four key objectives to realize the first complete solution for side-channel protected ML. First, we analyze the side-channel vulnerabilities in the various hardware blocks of an ML accelerator and assess the feasibility of model parameter extraction. Second, we design provably-secure gadgets, implement them on FPGA, and empirically validate possible countermeasures. Third, we add usability and flexibility to the solution—the ability to support multiple ML architectures via secure software APIs and compiler extensions on a RISC-V core. Fourth, we fabricate the final solution at Skywater 130nm node.

I. Introduction

Side channel attacks (SCA) are notoriously known to break the security of cryptographically secure algorithms [1] and leak secret data. SCAs exploit the correlation of the computed data on the physical properties of the device, such as the power draw, electromagnetic emanations, etc [1]. Since the discovery of differential power analysis (DPA) [1], the literature on SCA and defenses has matured significantly with two decades of academic research and industry adoption. However, the focus has always been on cryptographic implementations due to their strict confidentiality needs.

Machine learning (ML) has recently emerged as another application with confidentiality needs. ML models are expensive to develop, which makes them intellectual property (IP). ML IPs are the key drivers in the business of ML applications offered as a service (MLaaS). Thus, models should be protected against unauthorized access to their internals such as the weights, and biases (or parameters) in a neural network. Information about the internals of a model also aids in other potentially dangerous attacks such as adversarial attacks, model poisoning, and fault attacks [2]–[4]. Recent works have already shown the high potency of physical SCAs to extract the parameters of a neural network [5]. Given that the number of devices running ML is only going to increase in the coming years, the research on building side-channel defenses for ML hardware warrants urgent attention.

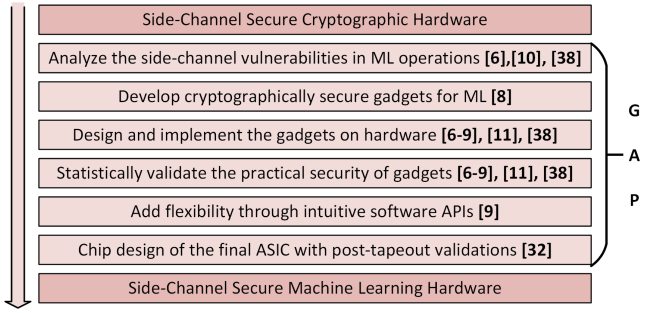

Fig. 1. The figure depicts the research gap that we address in our work starting all the way from assessing the side-channel vulnerabilities in ML hardware, to taping out a configurable side-channel secure ML ASIC.

We undertake the challenge to secure ML hardware against physical SCAs for the very first time through our work. We need to build a full-stack solution to solve this problem. ML was never designed for security, unlike cryptography. Therefore, we need to rethink the way we currently design ML applications. Fig. 1 highlights the research gap that exists between secure cryptography hardware and secure ML, and the various abstraction levels required to build a side-channel defense. We fill the gap through our seminal works, which span writing cryptographic proofs, designing hardware, developing compiler extensions, building a system-on-a-chip architecture, conducting side-channel validations, and taping out an ASIC.

We demonstrate the *first* successful DPA attack to extract the parameters of a neural network from an FPGA implementation. We develop the *first* side-channel defenses for neural networks. We design formally secure hardware gadgets to securely compute the common neural network operations like weighted summations, activations, maxpool, etc. We design and implement multiple secure neural network designs with varying performance, area, and security levels for both FPGA and ASIC targets. We use state-of-the-art tests like correlation power analysis (CPA), and test vector leakage assessment (TVLA) to validate the side-channel security of our implementations with millions of traces. Next, we add flexibility to our solution by coupling the secure neural network hardware

2023 IEEE International Test Conference (ITC) | 979-8-3503-4325-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ITC51656.2023.00035

2378-2250/23/$31.00 ©2023 IEEE
DOI 10.1109/ITC51656.2023.00035

with a RISC-V core. We propose a custom instruction set extension (ISE) to the RISC-V ISA to access the secure hardware gadgets and build a library of APIs that enable any user to securely perform neural network computations on our platform. Finally, we tape out our final solution targeting the Skywater 130nm node. We answer the following research questions through our contributions.

1) *To what extent is SCA exploitable in the low-level functions of ML, like weighted summation, activation function, etc., when implemented on hardware?*
   MaskedNet demonstrates the first successful power-based SCA on a hardware implementation of a neural network [6]. We show successful parameter extraction from a neural network on different ML operations and quantify the number of required queries for a successful attack.
2) *What are the potential security and computational bottlenecks when trying to leverage the existing cryptographic side-channel defenses to ML functions on hardware?*
   Our works MaskedNet and BoMaNet leverage the masking and hiding techniques to propose the first side-channel secure constructions of the neural network operations like weighted summation, activation function, etc. MaskedNet quantifies the *security bottlenecks* of using straightforward arithmetic masking on weighted summation, while BoMaNet explores a fully Boolean masking approach and quantifies the *computational bottlenecks* [7].
3) *What are the possible algorithmic transformations that can be incorporated into the ML algorithms to support an efficient adoption of side-channel defenses by design?*
   ModuloNet proposes a fundamentally new technique to perform neural network inference by incorporating modular arithmetic [8]. As observed from MaskedNet, and BoMaNet, the most efficient way to incorporate masking in neural network computations is to use modular arithmetic. We quantify the overheads of masking a neural network that uses modular arithmetic and show significant gains both in performance and area.
4) *How to flexibly support multiple ML architectures on custom-built and commercial accelerators while still maintaining side-channel resistance on hardware?*
   We develop a RISC-V-based coprocessor design that can securely process a neural network implemented in C/C++. We propose and implement a custom ISE to exercise the masking gadgets inside the coprocessor and use them to build a software library for secure ML functions [9].

A. Scope of this Paper and Organization

Due to space limitations, we have highlighted the key aspects of our works in this paper and skipped some details. We have organized this paper following the same theme of building the various abstraction levels from the top down. Section II presents the relevant background. Section III presents our attack from MaskedNet [6], which shows the vulnerability of neural network hardware to SCAs. Section IV presents two proofs from ModuloNET. Section V presents the details about the hardware design of the secure ML gadgets. Section VI describes the hardware software codesign to couple a RISC-V core to our secure neural network unit through custom instructions. Section VII describes the details of the ASIC tape out. Section VIII presents the area, and performance comparison of our works with other published works, and the side-channel validation results of our proposed designs. Section IX discusses the broader impact of our research in the education sector and industry. Additionally, we also published the first successful remote power attack to extract model parameters and a lightweight shuffling-based defense for neural networks, but skip those works in this manuscript [10], [11].

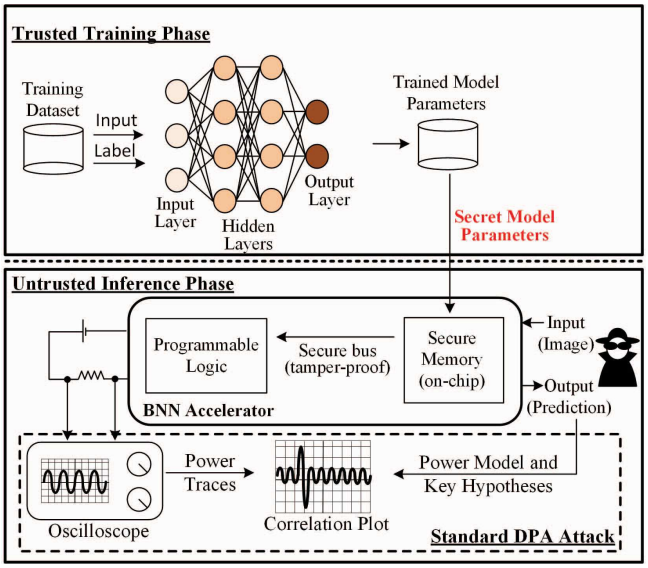

Fig. 2. DPA threat model applied to ML model stealing–the trained neural network is deployed to an edge device running in an untrusted environment.

II. Background

This section describes the commonly assumed threat model for side-channel attacks on ML hardware, hardware masking, the RISC-V framework, and neural network basics.

A. Threat of Physical Side-Channels for AI/ML

Numerous works have shown successful attacks to extract the model parameters from a device running ML inference [5], [6], [12]. We follow the same attack setup for our works as we depict in Fig. 2 and build defenses against it. The training happens securely and the computed parameters are programmed into a secure memory inside the edge device. The device then operates in an untrusted environment where an end user (adversary) can have physical access. *The adversary's goal is to learn the parameters of the neural network by conducting a DPA on the power traces captured during the inference computation on the device*. These parameters include weights and biases in fully connected layers and kernels in convolution layers. Adversary aims to steal the exact values of these parameters—known as the high-fidelity extraction of the model parameters [13]. Additionally, we also assume that the adversary knows the hyperparameters of the model either because it is public, or by using the techniques mentioned in prior works [5].

To mitigate side-channel leakage, we implement masking primitives that are proven first-order secure in the glitch-extended probing model in cryptography [14], [15]. We then validate the empirical security of our complete design using

both DPA and TVLA tests. We exclude invasive attacks such as clock glitching, or laser fault injection on the hardware. Fault attacks can be handled by a different layer of defense.

B. RISC-V ISA and Toolchain

RISC-V is an open-source instruction-set architecture with 47 base instructions [16]. Its modular design enables easy extensions over the base ISA. The encoding space of instructions is split into standard, reserved, and custom categories [16]. Any custom instruction should preferably use the encoding space allotted to the custom category because the standard space is already in use and the reserved space is kept for possible future standardization (see Fig. 8) The RISC-V cross-compiler (or toolchain) is publicly available with its source code. The relevant components for this work are the GCC, the Binutils, and the Newlib. Binutils contains the GNU assembler (`as`) and linker (`ld`). GCC is the GNU compiler for C, and Newlib provides the required low-level libraries for basic C routines like malloc, free, etc. Adding a new custom instruction requires modifying the source code of the toolchain and rebuilding it. The rebuilt toolchain can now compile a source code with the newly added custom instructions.

C. Hardware Masking

Masking is a common side-channel countermeasure. It splits the secret variable into multiple statistically independent and uniformly random shares and modifies the original algorithm to process these shares instead of the original secret and still maintain correct functionality. The power consumption is, therefore, decorrelated from the secret since the computations only happen on random shares. Based on whether the shares are split using exclusive-OR operation or modular addition, the scheme is respectively called Boolean or arithmetic masking. Multiple masking schemes have been proposed in the literature [17], [18]. We use domain-oriented masking (DOM) to mask Boolean functions because it is secure in the glitch-extended probing model and has a low randomness and area overhead [18]. Notably, this style of masking is also adopted in real-world products such as Google's OpenTitan [19].

D. Neural Networks Basics

Neural networks are a class of ML classifiers frequently used for classification problems. They consist of units called neurons that perform a weighted summation followed by a bias addition and a non-linear transformation. Multiple neurons are stacked together in layers that feed their results to the next layers. A fully connected (FC) layer has all its neurons connected to all the neurons of the previous layer. Another flavor of neural networks uses convolutional layers. The idea is to process smaller regions of the image and extract meaningful information using kernels before using the FC layers [20], [21]. The connection weights and kernel values are tuned during the training process and typically in floating point representation. However, to reduce the power and memory footprint for hardware implementations, quantized neural networks have been proposed that limit the precision of the parameters to fewer bits [22], with the extreme case being binarized neural networks (BNN) [23].

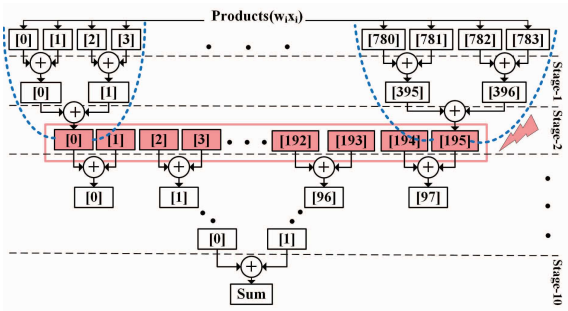

Fig. 3. Adder Tree used in HW Implementation. The figure shows the scenario where the 2nd stage registers(red) are targeted for DPA. This results in 16 possible key guesses corresponding to the 4 input pixels involved in the computation of each second stage register, grouped by the dotted blue line.

III. DPA on Parallel BNN Hardware

This section describes our attack on a BNN hardware implementation to extract secret weights. We use a completely parallel implementation of the neural network as the baseline. We assume the MNIST dataset for this implementation, which has 784 pixels per image. Thus, we design a pipelined adder tree of depth 10 to compute the complete sum in 10 cycles and achieve a throughput of 784 summations per cycle.

The pipeline registers of the adder tree store the intermediate weighted summations. Therefore, the value in these registers is directly correlated to the secret—model weights in our case. Figure 3 shows an example attack. Four possible values can be loaded in the output register $[0]$ of stage-1: $-[0]-[1]$, $-[0]+[1]$, $[0]-[1]$ and $[0]+[1]$ corresponding to the weights of (0,0), (0,1), (1,0) and (1,1), respectively[1]. Therefore, a DPA attack with known inputs (x_i) on stage-2 registers (storing $w_i x_i$ accumulations) can reveal 4 bits of the secret weights (w_i). The attack can target any stage of the adder tree but the number of possible weight combinations grows exponentially with depth. To aid the attack, we developed a cycle-accurate hamming-distance simulator for the adder tree pipeline and used it to mount a DPA attack.

Fig. 4 illustrates the result of the attack on stage-2 registers. There is a strong correlation between the correct key guess annotated with green and the power measurements crossing the 99.99% confidence threshold after 45k measurements. The attacker can successively extract the parameters for all the nodes in all the layers, starting from the first node and layer. The bias, in our design, is added after computing the final sum in the 10^{th} stage, before sending the result to the activation function. Therefore the adversary can attack this addition operation by creating a hypothesis for the bias. Alternatively, bias can be extracted by attacking the activation function output since the sign is correlated to the bias.

IV. Provably Secure Neural Network Gadgets

We present the security proof sketches for two of our proposed masked hardware gadgets from ModuloNET in this section: 1) masked weighted summation, and 2) Boolean-to-arithmetic conversion (B2A). We first define the two commonly used adversary models in literature viz. *t-probing security* and *glitch-extended t-probing security*.

[1]-1 is represented as 0 on BNN hardware for efficiency.

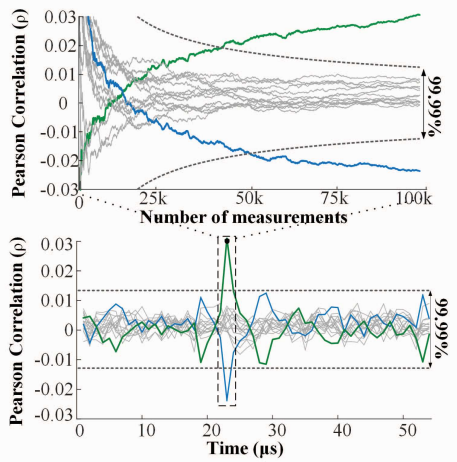

Fig. 4. Pearson Correlation Coefficient versus time and number of traces for DPA on weights. The lower plot shows a high correlation peak at the time of target computation, for the correct weight guess denoted in green. The upper plot shows that approximately 40k traces are needed to get a correlation of 99.99% for the correct guess. The confidence intervals are shown in dotted lines. The blue plot denotes the 2's complement of the correct weight guess.

Definition 4.1: t-probing security [24] A gadget G is *t-probing secure*, iff any arbitrary combination of every t-tuple wires in the gadget is independent of all secret variables.

Definition 4.2: Glitch-extended t-probing security [14], [15] 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 all secret variables.

Since we focus on masking ML-specific operations, we prove the first-order security of the ML-specific gadgets in the *glitch-extended probing model* [14] and provide *1-probing-secure* implementations for other gadgets. 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 $\mathcal{A}$. 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 we claim *first-order security*.

A. Weighted Summations:

Figure 5 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 4.1: G1 is glitch-extended t-probing secure given the secret variables as $w_{i,j}^{\{k\}}$.

Proof 4.1: (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.

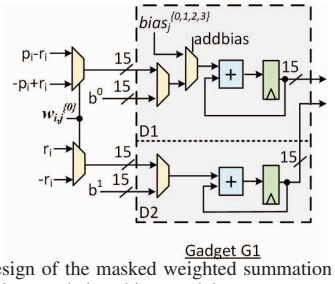

Fig. 5. Circuit design of the masked weighted summation gadget G1 proven secure in the glitch-extended probing model.

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 IV-B 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 [24]. Furthermore, although the weights are unmasked in the gadget, that is an issue with template attacks, not DPA.

B. 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 4.2: All the intermediate nets in G3 are independent of x.

Proof 4.2: (*Sketch*) The gadget pads the inputs before feeding them to the *Pipelined Golic's B2A* block (see Figure 6). 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

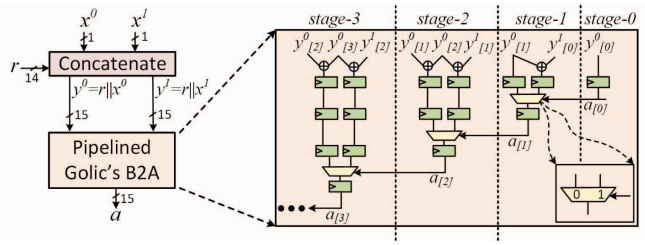

Fig. 6. Design of our proposed circuit to process the binary Boolean shares (x^0, x^1) to produce a such that $\bigoplus_i x^i = a + x^1$. The circuit concatenates the Boolean shares with 14 random bits and feeds them to a pipelined version of Golic's Boolean to arithmetic circuit.

$y^1 = r||x^1$; 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.[2].

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.

V. Hardware Design of the Masked Neural Network

We use a top-down approach to explain the fully masked ModuloNET hardware design shown in Fig. 7 (left). We start with an overview of the full design with various blocks and then explain the details of the masked weighted summation and activation function blocks. We skip the details of the other blocks for brevity. All the gadgets we develop are composable in the glitch-extended probing model, which automatically makes the full ModuloNET design also secure in that model.

[2]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.

A. Overview of the Full Design

We target an area-optimized BNN design that has a throughput of 1 summation per cycle; the input and hidden layers reuse the weighted summation unit. Our solution can extend to more parallel designs, but we make this decision to accommodate the full design on an FPGA and simultaneously analyze the side-channel leakage, which is the main focus of our work. We choose a common modulus (K) of 2^{15} for all the layer summations based on our offline characterization of the dataset [8]. Image BRAM stores the input pixels. The design sequentially reads one pixel per cycle, multiplies it with the respective weight, and then accumulates the weighted sum. The design performs this operation in a masked fashion, which is described in Section V-B. The masked summation is then fed to the masked activation function block that computes the Boolean shares of the activation result and stores them in the Layer BRAM. The design uses the XNOR-popcount technique to compute the weighted summations in the hidden layer. However, it needs to convert the Boolean shares of activation back to arithmetic shares before feeding them back to the adders to process the next layer. For output layer computation, the design feeds the masked summations to the masked output layer unit, which produces the masked classification result.

B. Arithmetic Masking of Weighted Summations

Fig. 7 shows the implementation of masked weighted summation using arithmetic masking because it is an arithmetic operation. The hardware reads p_i from the image BRAM, pads it to $log_2 K$-bits, and splits it into two arithmetic shares $p_i - r_i$ and r_i, where r_i is a $log_2 K$-bit fresh random mask. Next, the hardware directly and independently computes the weighted summations 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.

C. Masked Activation Function

The activation function receives two arithmetic shares and computes if their sum is greater 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. MaskedNet implemented a masked LUT-based ripple carry chain to produce the two Boolean shares of the MSB. Such a design is good for applications where the security requirements are low because glitches can happen inside the LUTs and eventually reveal the secret [25]. Prior literature suggests a number of alternatives such as Threshold Implementation [17], DOM [26], a synchronized variant of the Trichina's AND gate [7], etc., to perform masked operations with reduced glitches. We select DOM because it has low area and randomness requirements and provides security against glitches.

Figure 7 (right) shows our proposed novel design of the masked activation function using DOM-indep AND gates for 8-bit operands[3]. 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 sharings. The hardware needs to create Boolean

[3]We show an illustration with 8-bits but the actual design has $log_2 K$-bit operands.

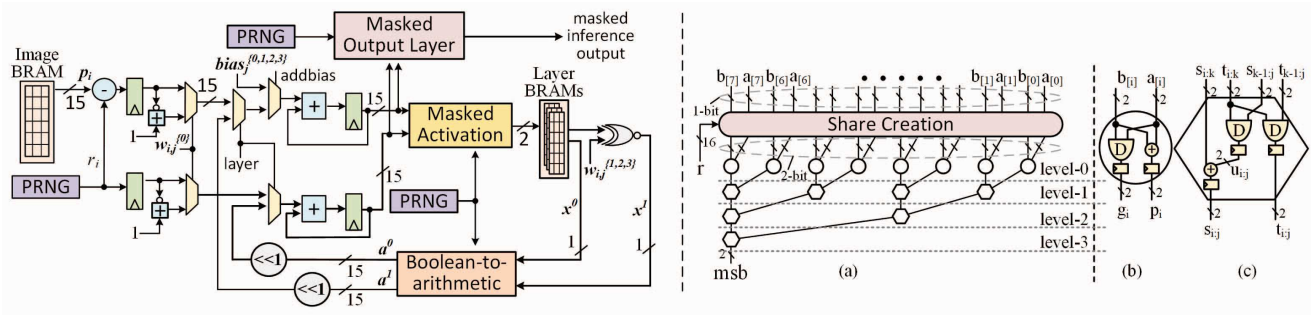

Fig. 7. Figure on the left shows the hardware design of the fully masked ModuloNET inference showing various top-level masked components. The weights $w_{i,j}^{\{k\}}$ and $bias_j^{\{k\}}$ are read from the respective block RAMs. The figure on the right shows the design of the masked activation function using a Kogge-Stone carry propagator with DOM gates labeled as D. Figure is not to scale.

shares of the input bits (of arithmetic shares) explicitly before feeding them into the DOM 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 [6], we implement the carry chain of a Kogge-Stone adder [27] because it has a logarithmic latency. 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.

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

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 $t_{i:i} = p_i$. The following equations illustrate the masking of the group generate and group propagate terms.

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

VI. Software-assisted Countermeasures for Flexibility

This section describes the hardware software codesign to add flexibility and usability to our side-channel secure neural network hardware. We again take the top-down approach to first explain the programming model exposed to the end user and then present the details of the implementation of the APIs using custom instructions, and coupling with the coprocessor.

A. Programming Model and Custom ISE

Instead of exposing the assembly-level custom instructions directly to the user, we expose C-based[4] APIs, which are easier to understand and debug. We provide the APIs as part of a shared library—the user needs to include the header file, follow the syntax of the functions, and link the library while generating the ELF. Our rationale behind the syntax of the APIs is to keep it close to existing ML frameworks. We develop an interface similar to the Sequential model of TensorFlow Keras [28]. The programming interface configures and executes each layer of the neural network sequentially similar to the Keras model of stacking layers.

A key advantage of our solution is exploiting the layer-wise reconfigurability of this software interface to activate/deactivate the masking of each layer. This helps to trade off security with performance. Selective masking has also been explored in cryptography—prior works propose masking only the first and last rounds of AES because those are the most vulnerable rounds in a DPA attack [29]. Neural networks are no different in this regard. Additionally, this also enables only protecting the parameters of the layers retrained using transfer learning and saves the power and clock cycles spent on unnecessary masking of layers with public parameters [30]. The library provides the following four APIs.

inst[4:2] / inst[6:5]	000	001	010	011	100	101	111
00	LOAD	LOADFP	**custom-0**	MISCMEM	OP-IMM	AUIPC	OPIMM32
01	STORE	STOREFP	custom-1	AMO	OP	LUI	OP32
10	MADD	MSUB	NMSUB	NMADD	OP-FP	OP-V	custom-2
11	BRANCH	JALR	reserved	JAL	SYSTEM	reserved	custom-3

(a)

funct7	rs2	rs1	minor	rd	major		
31:25	24:20	19:15	14:12	11:7	6:5	4:2	1:0

(b)

Fig. 8. Figures (a) and (b) respectively depict the overall opcode map for the RISC-V ISA and the encoding of an R-type instruction. Our design uses the major opcode of custom-0 (highlighted) to distinguish our proposed custom instructions from the base instructions, and the minor opcode to distinguish between each custom instruction.

```
void fetch_input(int* image, int num_images);
void input_layer(int* image, int num_images,
    int* weight0, int num_weight0, int* bias,
    int num_bias, int masking_enable);
void hidden_layer(int* weight, int
    num_weight, int masking_enable);
void output_layer(int* weight, int
    num_weight, int masking_enable);
```

We provide one API each for the input[5] and the output layer computations. We use the stacking approach for hidden

[4]Our approach is not fundamentally limited to a particular software language but we chose C given its suitability for embedded platforms.

[5]By input layer computation, we mean processing the input pixels to compute the first hidden layer.

layers: each call to the `hidden_layer(.)` API triggers a hidden layer computation where the hardware processes the previous layer results to compute the results of the next layer. We implement the APIs using inline assembly and the custom ISE. We describe the encoding of the custom instructions next.

B. Custom Instruction Set Extension

We choose the R-type instruction format shown in Fig. 8(a) for the custom instructions because some instructions require two source operands and a destination operand. Fig. 8(b) shows how the bits [6:2] are encoded for various categories of instructions. The base opcode encoding space has the two least significant bits [1:0] set to one. Since we only need 5 custom instructions, we choose the standard 30-bit base encoding space. We use the major opcode (bits [6:2]) to distinguish the custom instructions from the base instructions and the minor opcode (bits [14:12]) to distinguish between each custom instruction. We choose the *custom-0* space without loss of generality. The major opcode for this space is *00010*. The instruction names and operations are explained below.

1) `mnn.cfgwr rs1 rs2`. This instruction stores the pointer to a data structure (`rs1`) and its size (`rs2`) in a configuration register inside the coprocessor. It is used to store the pointers to the input pixels, the parameters, and the hyperparameters such as the layer sizes.
2) `mnn.<i/h/o>layer rs1`. These instructions have one source operand (`rs1`) controlling the enabling or disabling masking of the layers. Based on the opcode, the instructions `mnn.ilayer`, `mnn.hlayer`, and `mnn.olayer` respectively trigger the input, hidden, and output layer computations in the coprocessor.
3) `mnn.ifetch rs1 rs2`. This instruction fetches the required number of input pixels (`rs2`) from the host and stores them in the location pointed by `rs1` in hardware.

C. Codesigning the Hardware with Software

Fig. 9 shows the block diagram of our complete design. The important blocks are the RISC-V core (referred to as just *core* from hereon), the dual-ported memory accessible through a shared bus, and the coprocessor. The coprocessor further consists of the command decoder (CMD) and the secure neural network unit (SNNU). In the following sections, we describe the design details in a top-down fashion—how the design processes the high-level commands from the host to perform the secure inference on hardware eventually.

D. The RISC-V Integration

We select the open-source PicoRV32 RISC-V core [31] for this work because it has a low area footprint and provides the basic features we need for our solution. The design communicates to the host PC via UART. The host interface processes the received signals to generate specific commands using address mapping.

The host first loads the cross-compiled RISC-V binary to the memory. The binary contains the model parameters, hyperparameters, and custom instructions to perform the neural network inference. Next, the host sends the start signal to trigger the core to fetch instructions. The core has an in-built

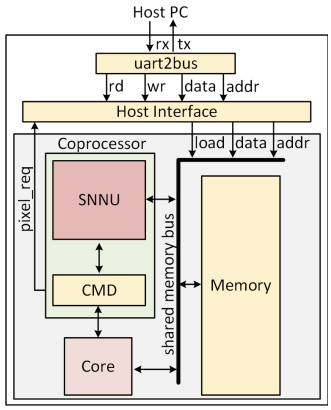

Fig. 9. The figure depicts major components of the proposed RISC-V based SoC. RISC-V and coprocessor units share the memory interface. CMD decodes the custom instructions sent on the PCPI interface and relays appropriate commands to the coprocessor to execute the required computations.

peripheral coprocessor interface (PCPI) to control a coprocessor. The CMD unit inside the coprocessor decodes the custom instructions discussed in Section VI-A and issues commands to the SNNU to perform the respective computations based on the decoded instruction. The following listing presents a typical user code to use our platform.

```
int main(){
  //declare and initialize pointers and
      variables
  init(image,w0,w1,w2,bias);
  fetch_input(image, ni);
  m_en=1;
  input_layer(image, ni, w0, nw0, bias, nb,
      m_en);
  hidden_layer(w1, nw1, m_en);
  output_layer(w2, nw2, m_en);
}
```

The core writes the input pixels, and parameters directly to the memory while executing the instructions. The coprocessor can directly read from the respective addresses during its computations because of shared memory access with the core. We choose to share the memory between the core and coprocessor instead of having distinct memories to avoid wasting cycles copying data from one memory to another.

VII. Physical Tape-out and Chip Fabrication

This section presents the details of the ASIC tapeout. We received the fabricated chips very close to this submission, and thus, this section is limited to only the pre-silicon tapeout information. Part of this information has been published at the GOMACTech [32]. Fig. 10 shows the final layout of our chip. We used the Skywater 130nm technology node for the tape out using the chipIgnite shuttle that is managed by eFabless Corporation. The shuttle uses a harness chip called Caravel as depicted in Fig. 10 (right). The chip consists of a management space and a user space. Management space consists of a RISC-V core, a small SRAM for the core, and some internal and external communication interfaces like Wishbone, UART, SPI, etc., to boot up the chip properly. The user design is instantiated inside the user space.

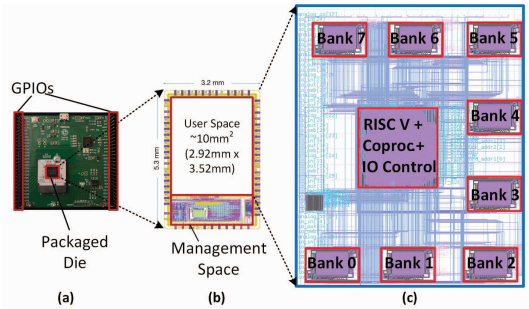

Fig. 10. The layout of the fabricated chip. The final solution fits into 10mm^2 with 8 SRAM banks for on-chip memory, RISC-V core, and our developed co-processors. This is placed into the Caravel architecture that allows communication with the outside world through UART.

We synthesized a single macro for our design depicted in Fig. 9. We place this macro at the center of the user space and surround it with eight SRAM macros of size 2kB to reduce congestion and routing delays. The Caravel harness provides direct access to 32 GPIOs from the user space. It also provides additional access to 128 logical ports that can be configured and driven by the RISC-V core in the management unit. We designate 2 GPIOs as the RX and TX pins of the UART protocol to communicate directly from host to the user space. We also add direct access to important control signals and registers inside the DUT via GPIOs to enable post-silicon debug. We use some of the logical ports from the management unit to configure the debug modes in the DUT. We use the management unit to only configure our design and not drive all the data from the host to reduce the communication latency, and to keep the DUT independent of the management unit.

VIII. Performance, Area, and Security Evaluations

This section presents the area and performance overheads of our proposed solutions, and how they compare to some other existing works. We also present the results of our side-channel evaluation on our hardware software codesign.

A. Area and Performance Comparison

Table I compares the area and latency of our solutions with prior works. It is difficult to make exact comparisons between the works because of the varying hyperparameters, parallelization modes, and implementation platforms (FPGA versus ASIC versus microcontroller). We try our best to provide the comparison for completeness. We first choose a standard hyperparameter set of 784 input nodes, one hidden layer of 512 nodes, and an output layer of 10 nodes for comparison; [33] already uses this configuration. All the works have a throughput of 1 summation/cycle[6]. Thus, we assume that the latency varies linearly with the total number of summations per inference—the sum of products of the number of nodes in two subsequent layers.

We scale the originally quoted latency (the Latency column) of the works to the expected latency of our chosen hyperparameter set (the Latency (N) column) for each work. The ASIC work [34] does not quote any latency numbers, thus, we assume it to be equal to the number of weighted summations. The results show that the ASIC and FPGA solutions have a comparable latency[7] of around 4×10^5, which is much lower than that of the microcontroller-based solution, as expected. However, while the microcontroller is programmable, the ASIC and FPGA are not. Our proposed hardware software codesign is almost as fast as the hardware solutions and provides the same programmability benefits as that of a microcontroller. Thus, it provides both programmability and high performance without sacrificing security.

[6]The ASIC solution [34] actually has a throughput of 8 summations/cycle but we assume only 1 PE instantiation for this comparison.

TABLE I
Area and Performance Comparison with Prior Works.

Work	Area (LUT+FF)	Latency (cycles)	Latency (N) (cycles)	Programmable
Our Work [7]	17457	2.94×10^6	4.2×10^5	No
Our Work [8]	10644	2.91×10^6	4.1×10^5	No
[33]	NA[1]	1.97×10^7	1.97×10^7	Yes
[34]	NA[2]	NA[3]	4×10^5	No
Our Work [9]	8778	10150	4.67×10^5	Yes

[1] microcontroller-based solution; no LUT/FF equivalents;
[2] ASIC solution; no LUT/FF equivalents;
[3] No latency numbers in the manuscript;

B. Side-Channel Validation

We use both DPA and test vector leakage assessment (TVLA) methods to perform side-channel validations [35]. We validate the empirical security with 1M power traces.

1) DPA Results

Fig. 11 shows the results of the DPA attack on the unmasked and masked implementations. We target the activation function for the attack, use the hamming distance power model[8], and hypothesize on the possible weights. We set 8 input pixels to be non-zero and hypothesize on the corresponding weights for those pixels. This reduces the number of hypotheses from 2^{64}, to 2^8. Fig. 11 (a) clearly shows a high correlation only for the correct hypothesis, at the exact point in time when the activation is computed. The leakage is statistically significant after 6000 measurements. Fig. 11 (b) shows the same attack on the masked implementation, which quantifiably fails even with 1M power traces. Note that the latency of the target operation in the masked design ($3.2\mu s$) is twice compared to that of the unmasked design ($1.6\mu s$) because the unmasked design uses both the datapaths to quickly compute the summations using a runtime reconfigurable hardware [9].

2) TVLA Results.

Since DPA is insufficient and atypical to evaluate side-channel security exhaustively, we also conduct the more generic TVLA test. We conduct TVLA on four masked neural network configurations C1 to C4. All the configurations have 1 input layer and 2 hidden layers of 64 nodes, and an output layer of 10 nodes. C1 and C2 respectively have all the layers unmasked and masked, C3 has only the second hidden layer unmasked, and C4 has only the output layer unmasked.

Fig. 12 (a) and (c) depict the overall power trace of a fully unmasked and fully masked configuration. We clearly observe the two hidden layer computations using simple power analysis as the two high amplitude bands; the bands have higher

[7]We refer to clock cycles. The actual latency might be much lower for an ASIC because of the high design frequency compared to FPGAs.
[8]The memory storing the activations is not reset between measurements.

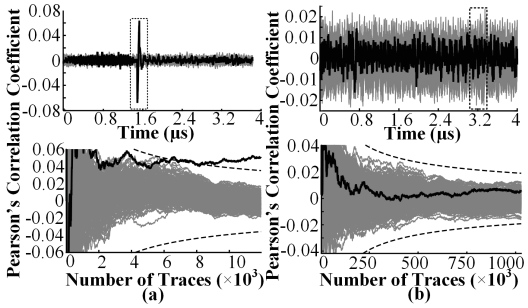

Fig. 11. Figure (a) top plot shows a high correlation peak only for the correct hypothesis in black; the bottom plot shows its evolution with the number of traces, which becomes statistically significant with the confidence of 99.99% (shown by the dotted lines) at 6000 traces. Figure (b) shows the same attack on the masked design, where we neither observe a high correlation peak for the correct hypothesis nor does the correlation become statistically significant before 1M traces demonstrating resistance to the DPA attack.

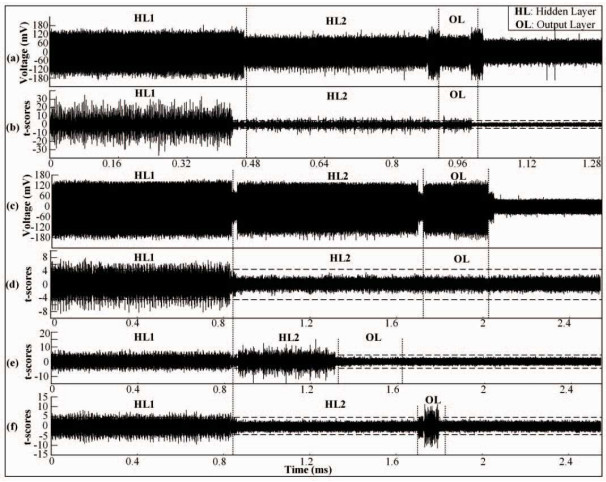

Fig. 12. Figures (a) and (c) show the mean power trace for the unmasked and masked designs. Figures (b), (d), (e), and (f) show the t-scores observed for the configurations C1-C4 in order. C1 and C2 are fully-unmasked and fully-masked designs, therefore, we observe t-scores higher than ± 4.5 (denoted by horizontal dashed lines) throughout the trace in (b), but not in (d). C3 has only HL2 unmasked, and C4 has only OL unmasked, and thus, the t-scores cross the threshold only during those regions in (e) and (f).

amplitude for the masked design compared to the unmasked design, which is expected because of extra activity due to PRNGs and masked datapaths. Also, the layer power activity is lower than that of the baseline processor activity in the unmasked design, which is why the power drops during the layer computations, and the power bumps are actually between the layer executions. The execution time for the masked design is twice that of the unmasked configuration due to the dual path reuse optimization in our design for the unmasked layers [9].

Fig. 12 (b) and (d) show the TVLA evaluations of C1 and C2. All the layers are unmasked in C1; thus, the t-scores cross the TVLA threshold of ± 4.5. C2 has all the layers masked, and therefore, we do not see the t-scores crossing the threshold throughout the execution except the input layer. This is because the design loads the pixels while computing the arithmetic shares, and that load operation results in the input correlations. This has been observed in prior works too [7], [8], and is verified by running an experiment with buffered arithmetic shares instead of generating them on-the-fly. We also verify our design with that approach. We used 1M traces in each fixed and random dataset to validate the side-channel security. (e) and (f) show the TVLA results for C3 and C4. We observe t-scores greater than the threshold in HL2 because the second hidden layer is unmasked. The HL2 execution time is halved with our optimization [9]. For C4, we observe the t-scores cross the threshold in OL because the output layer is unmasked. Thus, the design successfully masks/unmasks the layers based on user configuration.

IX. Broader Impact: Industry, Education Outreach, and Undergraduate Training

In this section, we present the research impact of our works in academia, and industry. We also talk about how we enabled students at the undergraduate level to conduct research in this domain, and successfully publish at top-tier venues.

Our works have created a significant impact in the literature in terms of spearheading the research on analyzing the side channel vulnerabilities in ML hardware applications and working towards building effective countermeasures against them. Our research has enabled numerous other works on extending the side-channel attacks on ML accelerators to multi-tenant FPGA platforms using remote power monitors (including our work) [10], [36], on using other masking schemes such as threshold implementation on ASIC targets, on using pure software masking on microcontrollers [33], [34], and also on extending our masking gadgets to include other physical defaults such as transitions [37]. Our research also achieved a technology transfer with Intel. Specifically, our designs were tested by Intel's Product Assurance and Security team (IPAS) on the Intel FPGAs, and the vulnerabilities were validated.

We have also introduced *underrepresented minority* undergraduate students to our research as part of the Research Experiences for Undergraduates (REU) program by the NSF. We worked with sophomore students at NC State University to mount a successful attack on a single neural network layer and extract the weights using DPA. We only used basic building blocks such as a breadboard, digital ICs, resistors, jumper cables, and the Analog Discovery platform. We also helped them to implement an effective low-cost countermeasure using the concept of wave-dynamic differential logic (WDDL). They were able to reduce the power side channel leakage by $15\times$ with the countermeasure [38]. Ashley Calhoun was the undergraduate lead student of this project and she published a first-authored paper. Ashley also traveled to the GLS-VLSI conference to present the paper in person. As a result of this experience, she has decided to pursue a graduate degree in electrical and computer engineering.

X. Conclusion

We conducted the first comprehensive research in the area of physical side-channel security for ML hardware. We demonstrate the vulnerability in parallel hardware for the first time, and we provided the first provably secure hardware gadgets to execute neural network operations securely. We also develop the first hardware-software codesign framework in the context of ML applications from a security standpoint. Our research had a significant influence on academia and a direct impact on

industry, along with a broader outreach to underrepresented undergraduate students. The vast variety of ML topologies and the nascent nature of this problem create a wide research space. Future works can explore the threat of other types of side channels, possible optimizations of solutions proposed in our works, or cover more types of neural network topologies under the threat of physical side channels.

REFERENCES

[1] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference Proceedings*, ser. Lecture Notes in Computer Science, vol. 1666. Springer, 1999, pp. 388–397.

[2] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2005, pp. 641–647.

[3] J. Steinhardt, P. W. Koh, and P. Liang, "Certified defenses for data poisoning attacks," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, 2017, pp. 3517–3529.

[4] C. Bozzato, R. Focardi, and F. Palmarini, "Shaping the glitch: Optimizing voltage fault injection attacks," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2019, no. 2, pp. 199–224, 2019.

[5] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. 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*. IEEE, 2020.

[6] A. Dubey, R. Cammarota, and A. Aysu, "MaskedNet: The first hardware inference engine aiming power side-channel protection," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST*. IEEE, 2020, pp. 197–208.

[7] ——, "BoMaNet: Boolean masking of an entire neural network," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD*. IEEE, 2020, pp. 51:1–51:9.

[8] A. Dubey, A. Ahmad, M. A. Pasha, R. Cammarota, and A. Aysu, "Modulonet: Neural networks meet modular arithmetic for efficient hardware masking," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2022, no. 1, pp. 506–556, 2022.

[9] A. Dubey, R. Cammarota, A. Varna, R. Kumar, and A. Aysu, "Hardware-software co-design for side-channel protected neural network inference," in *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2023, pp. 155–166.

[10] A. Dubey, E. Karabulut, A. Awad, and A. Aysu, "High-fidelity model extraction attacks via remote power monitors," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2022, pp. 328–331.

[11] A. Dubey, R. Cammarota, V. Suresh, and A. Aysu, "Guarding machine learning hardware against physical side-channel attacks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 3, 2022.

[12] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium, USENIX Security 2019*. USENIX Association, 2019, pp. 515–532.

[13] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[14] O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede, "Consolidating masking schemes," in *Advances in Cryptology - CRYPTO - 35th Annual Cryptology Conference Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 9215. Springer, 2015, pp. 764–783.

[15] S. Faust, V. Grosso, S. M. D. Pozo, C. Paglialonga, and F. Standaert, "Composable masking schemes in the presence of physical defaults & the robust probing model," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 89–120, 2018.

[16] K. A. Andrew Waterman, "The RISC-V Instruction Set Manual Volume I: Unprivileged ISA," https://github.com/riscv/riscv-isa-manual/releases, 2022.

[17] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *Information and Communications Security, 8th International Conference, ICICS, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4307. Springer, 2006, pp. 529–545.

[18] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," in *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS*, B. Bilgin, S. Nikova, and V. Rijmen, Eds. ACM, 2016, p. 3.

[19] OpenTitan. (2022) Opentitan. https://github.com/lowrisc/opentitan.

[20] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, Contour and Grouping in Computer Vision*, ser. Lecture Notes in Computer Science, vol. 1681. Springer, 1999, p. 319.

[21] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 826–834, 1983.

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[23] M. Courbariaux, Y. Bengio, and J.-P. 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*, ser. NIPS'15. MIT Press, 2015, p. 3123–3131.

[24] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," in *Annual International Cryptology Conference*. Springer, 2003, pp. 463–481.

[25] S. Mangard, T. Popp, and B. M. Gammel, "Side-channel leakage of masked cmos gates," in *Cryptographers' Track at the RSA Conference*. Springer, 2005, pp. 351–365.

[26] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," in *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, B. Bilgin, S. Nikova, and V. Rijmen, Eds. ACM, 2016, p. 3.

[27] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Computers*, vol. 22, no. 8, pp. 786–793, 1973.

[28] Google. (2022) The sequential model. https://www.tensorflow.org/guide/keras/sequential_model.

[29] E. Trichina, D. D. Seta, and L. Germani, "Simplified adaptive multiplicative masking for aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 187–197.

[30] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[31] C. Xenia Wolf. (2021) PicoRV32 - A Size-Optimized RISC-V CPU. https://github.com/YosysHQ/picorv32.

[32] A. Dubey, R. Cammarota, and A. Aysu, "Secure AI hardware by design: From cryptographic proofs to silicon tape-out," *GOMACTech*, 2023.

[33] K. Athanasiou, T. Wahl, A. A. Ding, and Y. Fei, "Masking feedforward neural networks against power analysis attacks," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 1, pp. 501–521, 2022.

[34] S. Maji, U. Banerjee, S. H. Fuller, and A. P. Chandrakasan, "A threshold-implementation-based neural-network accelerator securing model parameters and inputs against power side-channel attacks," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 518–520.

[35] B. J. Gilbert Goodwill, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST non-invasive attack testing workshop*, 2011, http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf.

[36] S. Tian, S. Moini, A. Wolnikowski, D. Holcomb, R. Tessier, and J. Szefer, "Remote power attacks on the versatile tensor accelerator in multi-tenant fpgas," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 242–246.

[37] M. Schmid and E. B. Kavun, "Analyzing modulonet against transition effects," in *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 2023, pp. 1–6.

[38] A. Calhoun, E. Ortega, F. Yaman, A. Dubey, and A. Aysu, "Hands-on teaching of hardware security for machine learning," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, 2022, pp. 455–461.