Stochastic Training for Side-Channel Resilient AI

Anuj Dubey*, Aydin Aysu*†
*Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, USA
Email: aanujdu@ncsu.edu, aaysu@ncsu.edu
†mithrilAI Corp., Cary, NC, USA
Email: mithrilai@gmail.com

***Abstract*—The confidentiality of trained AI models on edge devices is at risk from side-channel attacks exploiting power and electromagnetic emissions. This paper proposes a novel training methodology to enhance resilience against such threats by introducing randomized and interchangeable model configurations during inference. Experimental results on Google Coral Edge TPU show a reduction in side-channel leakage and a slower increase in t-scores over 20,000 traces, demonstrating robustness against adversarial observations. The defense maintains high accuracy, with about 1% degradation in most configurations, and requires no additional hardware or software changes, making it the only applicable solution for existing Edge TPUs.**

I. Introduction

Trained AI models have become invaluable assets, powering applications ranging from healthcare diagnostics and autonomous vehicles to personalized recommendations and financial forecasting [1]. These models represent significant investments of time, computational resources, and expertise. As a result, the confidentiality of trained AI models is critical not only to protect intellectual property but also to maintain the competitive advantage of organizations [2]. The leakage of trained AI models has implications beyond intellectual property theft. Stolen models can expose systems to other forms of attacks, such as adversarial learning, where attackers manipulate inputs to produce incorrect outputs, or fault injection attacks, which cause the system to behave unpredictably [3], [4]. These vulnerabilities undermine trust, compromise safety-critical applications, and weaken AI system reliability, making model protection essential for AI security [5].

Side-channel attacks pose a unique and increasingly relevant threat to AI model confidentiality, particularly for Edge AI devices deployed in uncontrolled environments [6]. Unlike centralized data centers, Edge AI devices operate closer to end users, providing real-time insights by running inferences on locally stored models. This accessibility makes them susceptible to side-channel attacks, where adversaries exploit physical emissions such as power consumption or electromagnetic (EM) signals to infer sensitive information [7]. The risk is especially pronounced in edge scenarios because these devices often lack the physical and software protections afforded to their cloud-based counterparts, amplifying the urgency of addressing this threat [8].

In recent years, significant efforts have been made to mitigate side-channel attacks targeting machine learning (ML) systems. Notably, some works have explored hardware-based defenses. MaskedNet introduces a hardware inference engine that employs secure hardware gadgets to protect against power side-channel attacks, necessitating specialized hardware design and implementation [9]. BoMaNet and ModuloNET extend these concepts to entire neural networks, further emphasizing the need for custom hardware solutions [10]–[12]. Conversely, others have focused on software-level countermeasures, pursuing the adaptation and implementation of such defenses through complex algorithmic transformations [13], [14]. However, these approaches are applicable exclusively to general-purpose processors or need custom instruction extensions as they require executing arithmetic/boolean operations not supported by native instructions of common edge AI accelerators.

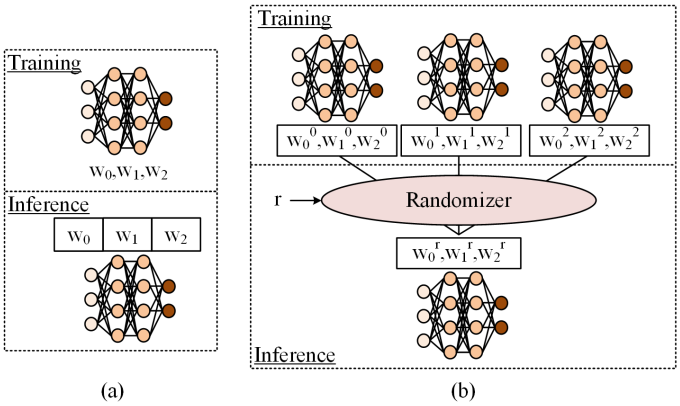

Fig. 1. The figure depicts regular neural network training on the left (a), and the proposed multi-model training for side-channel resistance on the right (b).

Therefore, a research gap remains in developing defenses that do not rely on specialized hardware or complex software instructions. This gap is particularly critical for existing edge AI devices, which already operate in the field and cannot leverage *any* of the prior defenses. Enhancing side-channel resilience on such devices represents a significant advancement, even if the achieved improvements may not offer the robust defenses seen in earlier works.

To counter these challenges, we propose a novel machine-learning training framework that can be retrofitted into existing Edge Tensor Processing Units (TPUs) and that makes side-channel attacks harder. Figure 1 outlines our approach. This figure compares the baseline machine learning (ML) training and inference pipeline (Part (a)) with our proposed defense mechanism (Part (b)). In the baseline configuration, a single ML model is trained, and the trained weights—denoted as W_0, W_1, W_2—are subsequently used for inference. Each weight W_i corresponds to a specific layer in the neural network, and the same trained model is consistently deployed

during inference. While effective for traditional use cases, this approach is vulnerable to side-channel attacks, as the deterministic behavior of the model enables adversaries to infer sensitive information through power or EM emissions.

By contrast, our proposed technique (Part (b)) introduces a randomized obfuscation layer to enhance resilience against side-channel threats. Instead of training a single model, multiple models are trained independently, with each producing a unique set of weights (W_0^i, W_1^i, W_2^i, where i indexes the model). During inference, a randomizer dynamically selects and combines weights from these models in a randomized manner, creating interchangeable layers across different neural network configurations. For instance, given three layers of neurons and three independently trained models, our approach enables $3 \times 3 = 9$ interchangeable layer combinations, significantly increasing the attack complexity and rendering the side-channel analysis less effective. This randomized, multi-model strategy ensures that each inference execution utilizes a unique configuration, thereby achieving robust obfuscation and enhanced security against side-channel attacks.

Our approach incorporates tailored defenses into the training process, creating models inherently resilient to information leakage through side channels. By modifying the training paradigm, our solution not only reduces the risk of side-channel attacks for edge deployments where previous defenses cannot be supported. Our contributions are as follows:

- We propose a novel training methodology that improves resistance to power/EM side-channel attacks.
- We implement the proposed training method on the Google Coral Edge TPU [15] and provide innovative techniques to address the compute limitations of the Edge TPU to retrofit our approach. The proposed method is integrated into the TensorFlow Lite (TFLite) framework.
- We evaluate the effectiveness of our approach through comprehensive experiments, demonstrating its resilience under practical attack scenarios. The result shows reduction in side-channel leakage and a marginal (≈1-2%) reduction of accuracy on an MNIST dataset.

II. Background

A. Side-Channel Attacks and Current Defense Limitations

Side-channel attacks exploit unintended information leakage from physical implementations of cryptographic systems, such as variations in power consumption, EM emissions, or timing information, to extract sensitive data [16]. These attacks pose a significant threat to cybersecurity because they can circumvent traditional cryptographic protections without needing to break the underlying algorithms. Recent years have shown that such attacks have spread to AI/ML systems to steal a trained model [6], [9], [17].

Power and EM side-channel attacks are particularly concerning for edge AI devices. In power analysis attacks, adversaries measure the power consumption of a device during operations to infer secret information. Similarly, EM attacks involve capturing EM emissions from a device to extract sensitive data. These attacks are feasible because computations produce data-dependent power consumption and EM radiation patterns, which can be analyzed to reveal secret information. The threat model assumes that an attacker aims stealing a trained AI model during inference and has physical proximity to the device to measure power consumption or EM emissions, making edge AI devices in uncontrolled environments especially vulnerable.

Several countermeasures have been developed to mitigate these techniques [9]–[11], [13], [14]. However, edge AI devices such as TPUs cannot support them because the native instructions readily available on such TPUs cannot implement the intricate compute units needed for these countermeasures relying on, e.g., hiding and shuffling. This calls for a fundamentally new approach to improve the security of existing edge AI devices.

B. The Software Ecosystem for Edge TPUs

Edge TPUs in the current version primarily supports inference, with small support for transfer learning only for the final layer. Since we assume the trained model to be already deployed on the device in our threat model, we will focus on secure inferencing for the scope of this paper.

Currently, there exist many software frameworks for ML model development. Tensorflow, and PyTorch are two leading ML frameworks to this end developed by Google, and Facebook, respectively [18], [19]. Both frameworks have an extension targeted specifically for low-end mobile/IoT devices called PyTorch Mobile, and TFLite [20], [21]. The Google Coral's Edge TPU currently only supports TFLite, and thus, we will limit our discussions to TFLite from hereon.

1) Tensorflow and TFLite

TensorFlow is a versatile open-source library designed for developing high-performance machine learning (ML) applications. Its architecture separates the frontend, where developers can write code in languages such as Python, C, or JavaScript, from the backend, which is implemented in highly optimized C++. This backend leverages specialized libraries like Eigen and cuDNN to accelerate numerical computations across diverse hardware platforms, including CPUs, GPUs, and TPUs. These optimizations make TensorFlow suitable for a wide range of ML tasks, from training to inference, in various computing environments.

TFLite is a lightweight extension of TensorFlow, specifically designed for mobile and embedded platforms with limited memory and computational resources. It optimizes models for inference by supporting specific data types such as 32-bit floating-point and 8-bit signed or unsigned integers, while excluding 16-bit floating-point numbers, ensuring compatibility with constrained devices. TFLite also supports hardware acceleration through its Delegate API, enabling integration with accelerators like Qualcomm, NVIDIA, and Google's Edge TPU. Additionally, it incorporates quantization to reduce model precision from 32- or 64-bit floating-point to lower-precision formats, significantly reducing computational demands while maintaining accuracy. For deployment on Edge TPUs, TFLite models require additional compilation using Google's Edge TPU Compiler, ensuring efficient execution tailored to the hardware's capabilities.

2) Edge TPU Compiler

The edgetpu-compiler is a command-line tool released by Google to compile TFLite files for the Edge-TPU. It is currently only supported on a Debian-based Linux operating system and a 64-bit x86 processor. Users can also upload their TFLite models on Google Colab, and use the web-based edgetpu-compiler. Since the compiled model is supposed to run on the Edge TPU, the compiler imposes further restrictions on the type of operations and data types to be used in the model to successfully map all the operations to the available hardware resources on the Edge TPU. We list the major restrictions below.

1) All tensors of the model should be quantized to 8 bits.
2) The sizes of tensors should be constant at compile-time.
3) The model should only use the supported operations listed in the official documentation.

The compiler supports most of the traditional neural network operations like matrix multiplication, ReLU, sigmoid, etc., which makes it suitable to run any deep feed-forward, or convolutional neural network efficiently. A critical aspect is that it lacks support for instructions that change the control flow, unlike a traditional processor. Thus, implementing conditional constructs like the if-else is not possible in the Edge TPU. This is because the design goal for the Edge TPU is to perform high-performance neural network inference, which does not have these complex constructs.

C. The Google Coral Edge TPU Hardware Architecture

The exact hardware architecture of our target edge TPU is currently not public. However, it is speculated to follow the same architecture as that of the cloud TPUs, with differences in the number of multiply-accumulate units used in the systolic array. The cloud TPU is mainly designed to parallelly execute a large number of neural network-specific operations with low latency. This is achieved by instantiating multiple compute units like adders, multipliers, activation functions, etc., that can simultaneously execute independent neural network operations. Since the TPU is primarily an arithmetic unit, it might lack the hardware support for complicated control flow instructions such as branch, or jump instructions. Given that the edge TPU is a scaled-down version of the cloud TPU, there is an even higher chance for the control flow hardware to be absent.

III. A Side-Channel Countermeasure for TPUs

To align our research with the current ML ecosystem, we tackle the challenge of providing side-channel security for existing commercial accelerators for the *first* time. This is a complex problem due to the constraints of commercial accelerators and the proprietary nature of their software stacks, which limit the implementation of traditional countermeasures.

As a case study, we select Google's Edge TPU, a purpose-built ASIC designed for high-performance inferencing on mobile and IoT devices [15]. We perform an in-depth analysis of its software ecosystem and on-chip hardware resources, focusing on vulnerabilities to physical side-channel attacks. Based on this analysis, we propose a novel countermeasure that achieves side-channel resistance without requiring modifications to the hardware or the proprietary compiler stack. Our approach leverages a fundamental property of machine learning algorithms, absent in cryptographic implementations, to serve as an entropy source for countermeasures. This underscores the potential of exploiting ML-specific characteristics to develop efficient and practical side-channel defenses.

Traditional power and EM side-channel defenses in cryptography rely on techniques like masking or hiding, which are not feasible here. The Edge TPU hardware cannot be modified, and its proprietary edgetpu-compiler generates executables directly, offering no access to intermediate instructions for countermeasures like instruction shuffling. Consequently, the only controllable aspect is the source code of the neural network deployed on the Edge TPU. To address this, we propose a novel technique that reduces side-channel leakage by working within these constraints, without requiring changes to the hardware or software stack.

A. Deep-Dive into Neural Network Training

1) Loss Functions

The loss function quantifies how distant is the prediction of the neural network from the correct prediction (also called *ground truth*)–it is high when the model accuracy is low, and low when the model accuracy is high. Mean squared error (MSE), and categorical cross-entropy loss are commonly used loss functions for regression, and classification tasks, respectively. Since we mainly focus on classification, we will discuss the categorical cross-entropy loss function $\mathtt{L}(\cdot)$ next. In the case of binary classification with just two labels 0 and 1, $\mathtt{L}(\cdot)$ is defined as follows.

$$\mathtt{L}(\hat{y_i}, y_i) = -y_i \times log(\hat{y_i}) + (1 - y_i) \times log(1 - \hat{y_i})$$

$$\hat{y_i} = model(x_i, \mathbf{W}, \mathbf{B})$$

y_i is the correct label (either 0 or 1), and $\hat{y_i}$ is the predicted probability by the model for label 1 [1]. Note that $\mathtt{L}$ is designed to be high if the model output $\hat{y_i}$ is low (close to 0) for an input corresponding to label 1 and vice versa. Due to the logarithmic variation, the loss is much higher for incorrect predictions, i.e., if the model predicts lower probabilities for label 1. The overall loss for a training step is actually the average loss $\frac{1}{m}\sum_{i=0}^{m-1} \mathtt{L}(\hat{y_i}, y_i)$ over all the m training samples x_i. For multiple classes, $\mathtt{L}(\cdot)$ can be generalized to the following equation.

$$\mathtt{L}(\hat{y_i}, y_i) = \sum_{c=0}^{N-1} (-y_{ic} \times log(\hat{y_{ic}}))$$

where c represents the output class, $\hat{y_{ic}}$ is a function of the input sample x_i, and the model weights W and biases B [2]. W and B are randomly initialized at the start of the training process. Training is essentially an optimization problem, where the objective is to minimize the loss function, i.e., to find the values of W and B such that the loss is minimized.

[1]The confidence scores at the output layer are often interpreted as probabilities for the respective classes.

[2]Model weights and biases vary during training, but fixed during inference.

Algorithm 1 Neural Network Training

```
1: procedure GRADIENT DESCENT(X,N,Y,α, E_p, n)
input: X,N,Y,α,E_p,n
output: W, B
2:     W ← random()
3:     B ← random()
4:     for i = 1 ··· E_p do
5:         Z_0 ← X
6:         for j = 1 ··· n do
7:             Z_{j+1} ← layer_j(Z_j, W_j, B_j)
8:         for j = 1 ··· n do
9:             W_j ← W_j − α Δ_{W_j} L(Z_n, Y)
```

a) Backpropagation

Algorithm 1 lists a typical sequence of steps executed during training. All the training samples and their corresponding labels are condensed into matrices X and Y, respectively. Each step consists of a *forward pass* and a *backward pass*. Lines 6-7 perform the forward pass, which involves evaluating the model function using the training samples X, weights W, and biases B. The algorithm computes the cumulative loss $\mathtt{L}$ using the evaluated output Z_n from the model and true labels Y.

Differentiation, or finding the derivative, is a well-known technique in calculus to identify the minimum or maximum of a function. In this context, we use the term *gradient* instead of derivative, as $\mathtt{L}$ is a multi-variable function. The gradient of a function f with respect to an input variable v, denoted as $\Delta_v f$, signifies the direction of the greatest change for that function. The training algorithm computes the gradients $\Delta_W L$ and $\Delta_B L$. To minimize the loss $\mathtt{L}$, the weights W and biases B are updated as $W - \alpha \times \Delta_W L$ and $B - \alpha \times \Delta_B L$, where α, known as the *learning rate*, determines the magnitude of the updates and requires initial tuning.

This iterative process, called *backpropagation* or the *backward pass*, updates W and B (as shown in Lines 8-9) to progressively reduce the loss. The training algorithm repeats these steps across multiple *epochs*, with E_p defining the total number of iterations. This overall process is popularly known as *gradient descent*, as the model iteratively "descends" along the loss function curve toward its minimum, improving accuracy with each epoch.

b) Stochastic Minibatch Gradient Descent

Performing the gradient descent over the entire dataset is too expensive computationally, and thus, prior works suggest an alternate approach called the *stochastic minibatch gradient descent*. The training samples are divided into smaller batches, with each step training on only one batch. While this introduces higher variation in model updates—since a single batch may not fully represent the dataset's features—it offers significant computational benefits with minimal impact on accuracy, making it widely favored in the literature. The technique also incorporates random sampling of batches, introducing stochasticity to eliminate biases from dataset ordering. Additionally, neural networks, as complex multivariable functions, can converge to different minima based on initial parameter values and sampled batches. Despite these variations, the resulting weights and biases often achieve similar accuracy. Next, we explore how this stochastic behavior in neural networks can be leveraged to design an effective side-channel defense.

Algorithm 2 Randomized Backpropagation

```
1: procedure GRADIENT DESCENT(X,N,Y,α, E_p, n, m)
input: X,N,Y,α,E_p,n
output: W, B
2:     for k = 1 ··· m do
3:         W^k ← random({1 ··· m})
4:         B^k ← random({1 ··· m})
5:     for i = 1 ··· E_p do
6:         Z_0 ← X
7:         for j = 1 ··· n do
8:             Z_{j+1} ← layer_j(Z_j, W_j^{r_j}, B_j^{r_j})
9:         for j = 1 ··· n do
10:            W_j^{r_j} ← W_j^{r_j} − α Δ_{W_j^{r_j}} L(Z_n, Y)
11:            B_j^{r_j} ← B_j^{r_j} − α Δ_{B_j^{r_j}} L(Z_n, Y)
```

B. Multi-Model Training

Figure 1 shows an overview of our defense. Figure 1 (a) depicts a regular training and inference phase. The model is trained over a dataset, and the trained model parameters are used during inference. Figure 1 (b) depicts our alternate proposal on training and inference, which can yield side-channel benefits without losing significant accuracy. During training, we propose to train multiple models with the same training dataset. Due to the stochastic nature of training, the trained model parameters will differ for each trained model. Once we get multiple trained models with different parameters, we propose two related solutions for the defense.

a) Secure forward pass

In the first solution, the model randomly selects the parameters of one trained model for each inference, akin to the shuffling defense used in cryptographic applications [22]. Prior work suggests shuffling the order of state byte evaluations in AES rounds to disrupt vertical side-channel attacks, which rely on computations occurring at the same time across measurements. Shuffling breaks this timing assumption, reducing vulnerability. Similarly, our defense processes inputs and intermediate values with a different parameter set for each execution, enhancing side-channel resistance. Increasing the number of trained models and random choices further strengthens security during inference.

b) Secure forward and backward pass.

One limitation of the first approach is that the amount of randomness during inference linearly depends on the number of trained models. We will see how to improve this in the second solution. The key difference in the second solution is that the random choice of parameters happens at the granularity of layers instead of models. Thus, during inference, the model now chooses the parameters for each layer randomly from the parameters of that layer from one of the trained models. This increases the number of possibilities during inference compared to the first solution. For instance, for a two-layer neural network, and two trained models, the inference can now create four model choices instead of two.

Our preliminary analysis showed that directly replacing the layer weights between multiple models can result in a significant accuracy loss. This is because the parameters of each layer in a trained model were trained only with respect to the parameters of the other layers in the same model. Randomly

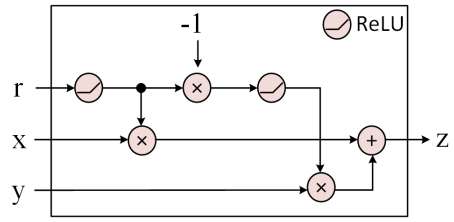

Fig. 2. The figure depicts how to construct an if condition using the ReLU function. The algorithm feeds r and $-r$ to two distinct ReLU units. The outputs are respectively multiplied by x and y. The final result is the sum of the products from the two branches.

altering parameters during inference can significantly deviate the model's behavior from its trained state. To address this, we propose integrating information about the parameters from other models into the training. Our new algorithm includes random layer selection during backpropagation to maintain consistency between training and inference stages.

Algorithm 2 outlines the randomized backpropagation process. The number of parameter choices for each layer corresponds to the number of models, denoted by m. At the start of each epoch, the algorithm uniformly and randomly samples parameters for each layer from $1 \cdots m$ (lines 2-4), with superscripts distinguishing parameters of different models. To preserve model behavior, distinct layer choices are sampled for each training sample x_i in X. The algorithm then performs a forward pass using the selected parameters for each layer and computes the final output Z_n.

Next, the algorithm propagates gradients only to the selected parameters for the corresponding training sample. By iterating over multiple epochs, the parameters across models are continuously updated in relation to each other. This effectively trains a larger model that uses only a subset of itself during inference, maintaining reasonable accuracy. However, implementing conditional selection in the forward pass remains a challenge, as detailed in the next subsection.

C. Key Challenge: Conditional Statements on Edge TPU

Having established a secure training and inference algorithm, we now demonstrate its implementation on the Edge TPU. While training occurs offline, all trained model parameters, along with the network architecture, are packaged into a single model for deployment on the Edge TPU. However, implementing inference with random parameter selection presents significant challenges.

As shown in Figure 1, implementing the randomizer involves conditional instructions. The network must select layer parameters from the random input and execute computations using them. While the latter is straightforward—requiring only weighted summations and activation function evaluations—the first step, involving control flow operations, is non-trivial on the Edge TPU. As discussed in Section II-C, the TPU hardware likely lacks support for control flow instructions. To test this, we compiled programs with constructs such as `if-else` and `tf.cond` from the TensorFlow API. These constructs work on a host CPU but fail to compile with the edgetpu-compiler, showing the Edge TPU's control flow limitations.

To implement our defense on the Edge TPU without explicit control flow statements, we adopted a fundamentally different approach. After examining all supported Edge TPU operations, we found no direct equivalent of a conditional statement. Consequently, we constructed one using existing operations—a function that takes three inputs (a, b, r) and outputs either a or b, depending on the value of r.

The rectified linear unit (ReLU) function proved particularly useful for this purpose. It is defined as:

$$\texttt{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

The definition of the ReLU function inherently contains an `if` condition, selecting either the input x or zero based on the sign of x. By randomly choosing r between -1 and $+1$ and multiplying it with x, the result is x 50% of the time. While this partially meets our requirement of selecting a number based on random input, it also outputs zero 50% of the time.

To fully achieve the desired functionality, we multiply r by -1 and then multiply the result with y. This creates two mutually exclusive branches: one branch outputs x and the other outputs zero, or vice versa with y. Adding the outputs of both branches gives x when $r = +1$ and y when $r = -1$.

Figure 2 demonstrates the construction of an `if-else` statement using the ReLU operation. Leveraging this approach, we design a novel function composed solely of ReLU operations that effectively replicates an `if-else` construct. This function enables the selection of different parameters from each model during inference on the Edge TPU.

IV. Implementation Results

We next discuss our experimental setup, the tests and results for the validation of side-channel leakage reduction on an edge TPU, and the impact of the proposed training on accuracy.

A. Measurement Setup

Our target board is Google Coral's Dev Board, which is a development board that hosts the edge TPU on a removable system on module (SoM) [23]. We choose EM side channels for this evaluation to precisely capture the activity directly from the leaky points on the TPU. We use Riscure's high-sensitivity EM probe to capture the EM emanations from the TPU [24]. We also use the EM Probe Station provided by Riscure which uses an XYZ table along with the EM probe to spatially scan the chip for high EM activity. This is required because the floorplan of the edge TPU has not been released. Our setup first performs a spatial scan of the entire TPU chip while running inferences, and captures the EM activity at each point. Then we use a bandpass filter to find the subset of points that correspond to an EM activity at 500 MHz, which is the operating frequency of the TPU core. We fix the probe position to this location and conduct the rest of the validation.

B. Side Channel Validation

Without loss of generality, we evaluate two MLPs for the MNIST dataset with configurations 784-10, and 784-100-10, and call them N1 and N2. For both networks, we conduct three sets of experiments I, II, and III. Experiment I corresponds to a network without any countermeasure. Experiment II

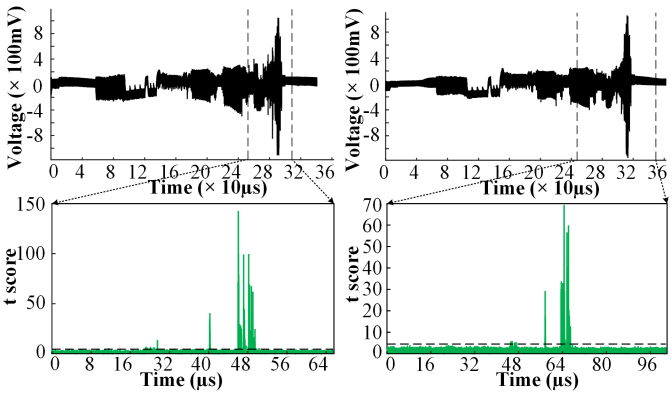

Fig. 3. The figure shows averaged power consumption and TVLA results from our experiments. We observe a reduction of 50% in the t-scores for III compared to II.

corresponds to the network with the countermeasure blocks present, but the randomness is disabled. Essentially all the random values are fixed to -1 in every measurement. This setting is expected to be equivalent when the defenses are turned off. Experiment III corresponds to the network with countermeasures enabled by actually changing the randomness for each measurement. Our goal is to quantify the amount of side-channel leakage in 1) a vanilla neural network with no defense, 2) a network with additional logic for defense but disabled randomness, and 3) the final solution with defense enabled. We use TVLA[3] for assessment.

Figure 3 shows our results on N1. We observe that the t-scores cross the threshold of ± 4.5 for all three experiments. However, the magnitude of t-scores decreases from I to III. The t-scores are highest in I because of the lack of any defense; the image is processed directly with the same parameters for each inference causing a high information leakage. The t-scores are lower in II compared to I because the noise caused by the additional logic of the defense decreases the signal-to-noise ratio in the side channel measurements. Although the defense is disabled, it still imparts some protection. The t-scores reduce by approximately $2\times$ in III. This is because of the active defense. The new inference algorithm chooses a different set of parameters for each inference based on the random input. This reduces side-channel leakage by lowering the likelihood of the same parameters being processed in each measurement.

Figure 4 shows the variation of the t-scores with the increasing number of measurements. We conduct a TVLA with 20,000 measurements, thus, the fixed and random sets contain approximately 10,000 traces each. As the setup captures more measurements, we observe an increasing trend in the t-scores for both II and III. However, the increase is much more rapid in II compared to III. This implies that with the defense enabled, the increase in the side channel leakage with a number of measurements is substantially lower that the implementation with the defense disabled.

[3]Test Vector Leakage Assessment (TVLA) is a statistical method widely used in side-channel analysis to evaluate whether a system exhibits data-dependent leakage. It compares side-channel traces collected under two different conditions, such as fixed and random inputs, using statistical metrics like Welch's t-test. A t-score exceeding a threshold indicates significant leakage, suggesting that an attacker could potentially exploit the side-channel information. TVLA is a reliable and efficient tool for assessing the security of hardware against side-channel vulnerabilities.

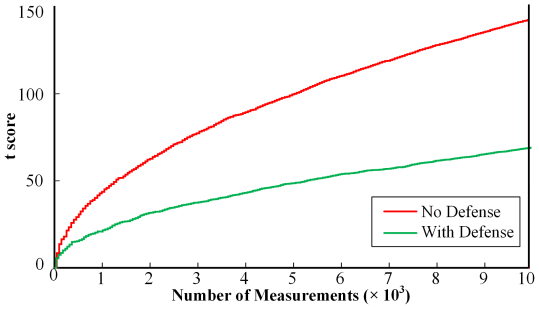

Fig. 4. The figure shows the evolution of t-scores with the number of traces in experiment II (red) compared to III (green).

TABLE I
ACCURACY COMPARISON OF THE MODELS WITH DEFENSE

Architecture	Baseline	Modelwise	Layerwise
784-10	91.98%	91.24%	91.52%
784-100-10	97.35%	97.29%	95.89%
784-100-100-10	97.68%	97.47%	96.30%

C. Accuracy Evaluations

We evaluated multiple models to determine whether the proposed defense impacts accuracy. Table I summarizes the results, using the MNIST dataset for all evaluations. The experiments were conducted on the same MLP networks used in side-channel evaluation. We also added a third one with another layer of 100 neurons to identify if the proposed training would have a different impact as the layer complexity increases. The results indicate that the accuracy of the models with our defense remains comparable to the baseline model without defense. While the layerwise models show a slight drop in accuracy, this is attributed to splitting the same training dataset across multiple models, resulting in less data for each individual model. This issue can be mitigated by increasing the number of training epochs to provide more iterations for learning.

V. CONCLUSION

In this work, we proposed a novel training methodology to enhance the resilience of edge AI devices against power and EM side-channel attacks. By leveraging the stochastic nature of neural network training, our approach dynamically creates interchangeable model configurations during inference, significantly complicating side-channel analysis. Experimental results demonstrated a reduction in side-channel leakage and a slower increase in t-scores over extended measurements while maintaining about 1% accuracy degradation compared to baseline models. These findings highlight the practicality of our method, which requires no hardware or software modifications, making it particularly suited for resource-constrained edge devices. This solution paves the way for robust, scalable defenses against emerging threats in the rapidly evolving landscape of AI security.

ACKNOWLEDGMENTS

This project is supported in part by NSF under Grants No. 1943245 and 2333126.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[2] Rambus, "Side-channel attacks target machine learning (ml) models," *Rambus Blog*, 2019.

[3] N. Papernot, P. McDaniel, and A. Swami, "The limitations of deep learning in adversarial settings," *Proceedings of the IEEE European Symposium on Security and Privacy*, pp. 372–387, 2016.

[4] S. Tajik and F. Ganji, "Artificial neural networks and fault injection attacks," *arXiv preprint arXiv:2008.07072*, 2020.

[5] X. Wang, J. Li, X. Kuang, Y.-a. Tan, and J. Li, "The security of machine learning in an adversarial setting: A survey," *J. Parallel Distrib. Comput.*, vol. 130, no. C, p. 12–23, Aug. 2019. [Online]. Available: https://doi.org/10.1016/j.jpdc.2019.03.003

[6] L. B. et al., "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel," in *USENIX Security '19*, 2019.

[7] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. 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*, 2018, pp. 393–406.

[8] X. Jin, C. Katsis, F. Sang, J. Sun, A. Kundu, and R. Kompella, "Edge security: Challenges and issues," *arXiv preprint arXiv:2206.07164*, 2022.

[9] 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 2020, San Jose, CA, USA, December 7-11, 2020*. IEEE, 2020, pp. 197–208.

[10] A. Dubey, R. Cammarota, and A. 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*. IEEE, 2020, pp. 51:1–51:9. [Online]. Available: https://doi.org/10.1145/3400302.3415649

[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] 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.

[13] 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.

[14] A. Dubey, R. Cammarota, A. Varna, R. Kumar, and A. Aysu, "Hardware-software co-design for side-channel protected neural network inference," in *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2023, San Jose, CA, USA, May 1-4, 2023*. IEEE, 2023, pp. 155–166. [Online]. Available: https://doi.org/10.1109/HOST55118.2023.10133716

[15] Google, "Edge TPU," 2022, https://cloud.google.com/edge-tpu.

[16] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397.

[17] 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)*, 2022, pp. 328–331.

[18] Google, "Tensorflow," 2022, https://www.tensorflow.org/.

[19] Meta, "Pytorch," 2022, https://pytorch.org/.

[20] Meta, "Pytorch mobile," 2022, https://pytorch.org/mobile/home/.

[21] Google, "Tensorflow lite," 2022, https://www.tensorflow.org/lite.

[22] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F. Standaert, "Shuffling against side-channel attacks: A comprehensive study with cautionary note," in *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, ser. Lecture Notes in Computer Science, X. Wang and K. Sako, Eds., vol. 7658. Springer, 2012, pp. 740–757. [Online]. Available: https://doi.org/10.1007/978-3-642-34961-4_44

[23] Google, "Dev board," 2020, https://coral.ai/products/dev-board.

[24] Riscure, "High precision em probe," 2020, https://getquote.riscure.com/en/quote/2101073/high-precision-em-probe.htm.