Using Undervolting as an on-Device Defense Against Adversarial Machine Learning Attacks

Saikat Majumdar, Mohammad Hossein Samavatian, Kristin Barber, Radu Teodorescu
Department of Computer Science and Engineering
The Ohio State University, Columbus, OH, USA
{majumdar.42, samavatian.1, barber.245, teodorescu.1}@osu.edu

***Abstract*—Deep neural network (DNN) classifiers are powerful tools that drive a broad spectrum of important applications, from image recognition to autonomous vehicles. Unfortunately, DNNs are known to be vulnerable to adversarial attacks that affect virtually all state-of-the-art models. These attacks make small imperceptible modifications to inputs that are sufficient to induce the DNNs to produce the wrong classification.**

In this paper we propose a novel, lightweight adversarial correction and/or detection mechanism for image classifiers that relies on undervolting (running a chip at a voltage that is slightly below its safe margin). We propose using controlled undervolting of the chip running the inference process in order to introduce a limited number of compute errors. We show that these errors disrupt the adversarial input in a way that can be used either to correct the classification or detect the input as adversarial. We evaluate the proposed solution in an FPGA design and through software simulation. We evaluate 10 attacks and show average detection rates of 77% and 90% on two popular DNNs.

***Index Terms*—undervolting, machine learning, defense**

I. Introduction

Deep neural networks (DNNs) are emerging as the backbone of an increasing number of diverse applications. Some of these applications benefit from the deployment of sophisticated DNN models into so-called edge devices such as smartphones, smart home devices, autonomous driving systems, healthcare and many others [6], [32], [38]. Some of these applications, such as autonomous driving, are safety critical and their failure can endanger lives. Unfortunately, DNNs are known to be vulnerable to a variety of security threats. One of these threats is the so-called "adversarial attack" against DNN classifiers. The goal of these attacks is to induce the DNN to misclassify an input that the attacker controls, into the wrong class. This is achieved by slightly altering the input to the classifier model in a way that induces it to produce erroneous an erroneous result. For example, the image of a *STOP* sign can be slightly modified to cause a road sign recognition model to classify it as a *YIELD* sign in spite of those changes being imperceptible to the human eye.

A broad spectrum of prior work has demonstrated successful attacks on image classifiers and other computer vision applications [8], [10], [15], [27], [35], [39], [40], [45]. Prior work has also shown that virtually all classifiers are vulnerable including the popular ResNet [18], AlexNet [24], VGG [51]. The most recent, strongest attacks [8], [10] achieve reliable misclassification with changes to the input images that are small enough to be undetectable by casual observation.

In response to these attacks, prior work has proposed a suite a defenses [7], [12], [33], [34], [44], [58]. These defenses generally rely either on model retraining, which are not easily generalizable, or online, inference-time defenses, which are mostly software-based and have very high overheads. Only a few hardware-accelerated defenses have been proposed to date. DNNGuard [55] is one example that relies on a separate dedicated classifier for adversarial detection, which also comes at a high cost.

Prior work such as [20] has shown that adding some amount of random noise to inputs can help DNNs correctly classify adversarial inputs. Recent work by Guesmi et al. [17] proposed using approximate computation to introduce a controlled number of error in the DNN inference to correct some of the adversarial input into valid classifications. The observation that these and other prior work has made is that some amount of alteration to the model or input tends to affect adversarial inputs more than benign ones.

In this paper we propose a new, lightweight adversarial correction and/or detection mechanism for image classifiers that relies on the compute errors introduced by undervolting (running a chip at a voltage that is slightly below its safe margin). Prior work has shown that DNNs tend to be resilient to the errors introduced by undervolting [49], [50] in an FPGA. In this work we perform a thorough characterization of the effects of undervolting errors on classification of both adversarial and benign inputs. We show that errors during inference lead classifiers to assign a different class to adversarial images than would be assigned in the error-free inference. The new classification is either the correct class for the input (before the attack) or is a third class that is neither the original or the intended target of the attacker. We also show that undervolting is much less likely to lead to the misclassification of benign (unaltered) images. Based on these empirical observations, we propose a lightweight, energy efficient defense mechanism targeted at deployment on energy constrained edge devices.

Our proposed solution uses undervolting to achieve two goals: (1) introduce random errors caused by low-voltage operation into the DNN inference execution and (2) save energy by operating at the lower voltage.

This work was funded in part by the National Science Foundation under awards CCF-2028944 and CCF-1629392 and by the Air Force Research Laboratory under the Assured and Trusted Microelectronics Solutions award FA8650-20-C-1719.

Prior work has shown that errors introduced by undervolting are both random (they occur in random locations in different chips) and repeatable (they occur in the same location within a chip) [3], [41]. The random distribution makes the defense unique to each chip, making it harder for an attacker to generate a successful general attack. The repeatably of the errors allows for system control over the approximate error rate that a chip is likely to encounter, allowing chips to continue execution without system crashes, in the presence of occasional localized errors. Prior work has used such controlled errors in security, for example, as a mechanism for hardware-backed authentication [4] or to attack SGX secure enclaves in Intel processors [41]. We propose using the same controlled errors to inject noise in the processor while running a DNN classifier.

We investigate two possible designs: one focused on adversarial correction, the other focused on detection. In the correction-based mechanism the system lowers supply voltage to a pre-determined set point, known to induce a certain number of errors, while the system is running the inference process. We show that this very simple approach leads the classifier to correctly classify an average of 50-55% of adversarial inputs, generated with multiple attacks. The second mechanism, focused on adversarial detection, compares the classification outcome of an error-free reference run with the undervolted run. If they are different, the input is classified as adversarial. Detection rate averages across a range of attacks are 77% and 90% on DenseNet and VGG16, respectively.

We also developed a lightweight on-device training process that fine tunes the model to the error profile on each device. This process improves the classification accuracy for benign (non-adversarial) inputs in the presence of undervolting errors, while keeping the correction/detection rates of adversarials largely unchanged. Finally, we show that the proposed defense is resilient against attacks that have knowledge of the defense and attempt to circumvent it by training on a model with random errors or a fixed error distribution.

We evaluate the proposed system with an FPGA implementation of a DNN accelerator (which we undervolt with an external controller) as well as an error-model based software implementation that allows us to measure the impact of multiple error distributions and rates. We evaluate 10 attacks or attack variants on two popular deep convolutional neural networks, VGG16 [51] and DenseNet [21], on the ImageNet [25] and CIFAR [23] datasets.

This paper makes the following contributions:

- The first work we are aware of to propose using undervolting errors as a defense against ML adversarial attacks.
- A lightweight on-device training mechanism to improve accuracy of benign inputs under errors.
- A thorough study on the effects of different error rates and distributions on the adversarial detection/correction rates as well as the effects on accuracy of benign classification.
- Evaluates the proposed system on a FPGA platform and using a software-based model.

The rest of this paper is organized as follows: Section II presents background and related work. Section III details the threat model. Section IV presents the details of the proposed defenses. Sections V and VI present the evaluation and Section VII concludes.

II. Background and Related Work

A. Undervolting

Voltage underscaling is a common power saving approach, whereby reducing the supply voltage causes a significant reduction in power consumption. A large body of work [3], [13], [29] has explored approaches to dynamically reduce voltage margins to current operating conditions, in a process generally referred to as ”undervolting”. For example, the well-known Razor [13] design dynamically lowers supply voltage until occasional timing errors occur. Additional latches running on a delayed clock are used on the vulnerable paths to detect and recover from these errors. Other work by Lefurgy et al. [29] uses hardware critical path monitors specially built into the chip to detect when the processor approaches its timing margin as a result of undervolting. Salami et al. [49], [50] have characterized extensively the behavior of FPGAs running a DNN accelerator with undervolting. They also provide experimental analysis of undervolting below the safe voltage, which leads to observable faults that manifest as bit flips. Similar studies have been performed by undervolting various system components in CPUs [3], [42], [22], [5], GPUs [30], DRAMs [9] focusing primarily on fault characterization, voltage guardband analysis and fault mitigation.

B. Adversarial Machine Learning Attacks

Szegedy et al. in [53] showed that small perturbations to the inputs can force machine learning models to misclassify. Most attacks follow the same general approach. Let f be a probabilistic classifier with logits f_y and let $F(x) = \arg\max_y f_y(x)$. The goal of the adversary is to find an L^p-norm bounded perturbation such that $\Delta x \in B^p_\epsilon(0) := \{\Delta : ||\Delta||_p \leq \epsilon\}$, where ϵ controls the attack strength, such that the perturbed sample $x + \Delta x$ gets misclassified by the classifier $F(x)$. The attacker's goal can be formulated as:

$$\text{given } x, \text{find } x' \quad \text{s.t. } ||x' - x||_p \leq \epsilon_{max} \text{ and } F(x') \neq y \quad (1)$$

In the case of *targeted* attacks, the adversary's goal is for $\mathbf{x}'$ to be misclassified as $F(x') = t$ with t being a class different than y.

FGSM [15] is a fast, simple and efficient method for generating adversarial attacks using the L_∞ norm that measures the gradient of the loss with respect to the input data, then adjusts the input data to maximize the loss. The **Carlini-Wagner (CW)** attack [8] finds adversarial examples with considerably smaller perturbations and higher accuracy. CW has variants for the L_0 and L_∞ distortion metrics. CW-L_0 uses the CW-L_2 attack to identify pixels which do not contribute much to the classifier output and freezes those pixel values. CW-L_∞ revises the objective function to limit perturbations to be less than a threshold, iteratively searching for the smallest possible threshold. The **EAD** attack [10] formulates the objective function in terms of regularized elastic-nets, incorporating the L_1 distance metric for distortion, where elastic-net regularization is a linear mixture of L_1 and L_2 penalty functions. EAD has two variants, where the optimization process can be tuned by two different decision rules: *EN*, the elastic-net loss or L_1, the least L_1 distortion. **DeepFool** [40] proposed a method to generate adversarial inputs by searching for the closest distance from the benign image to the decision boundary of the target image. We test our defense against multiple variants of the aforementioned attacks.

C. Defensive Techniques

Current defense methods against adversarial attacks can be divided into the following broad categories:

1. Making the network itself robust [56], [43], [35], [26] by hardening the model, for example, by using *Adversarial Training*. The training set can be augmented with adversarial samples so that the network learns to classify adversarial samples correctly. In general, this class of defenses relies on known attacks and is difficult to generalize [43].

2. Transforming the input samples [31], [52], [58] which pre-processes the model inputs, such as by encoding or filtering, to denoise the adversarial perturbation.

Prior work has also attempted to detect adversarial inputs. Approaches proposed by [58] aim to "squeeze" an input image and [12] applies weighted dropouts to neurons, to detect an adversarial sample. Grosse et al. [16] proposed a statistical test using maximum mean discrepancy and suggests the energy distance as the statistical distance measure for detecting adversarials. Other approaches like [16], [37] suggest training a detector using adversarial examples.

In general, methods that rely on re-training are not generalizable. Detection approaches are more successful, but their overhead is generally very high. For instance, Feature Squeezing [58] has a 3-4× overhead because it relies on 4 models for detection. Our proposed solution is lightweight and highly effective at detecting a wide variety of adversarial attacks.

III. Threat Model

This paper assumes adversaries that have complete access to the DNN model, with full knowledge of the architecture and parameters, and are able train their attacks accordingly. We assume an adversary that is able to directly deliver inputs to the DNN classifier, such as in [14], [46], [47] where attackers generate visual adversarial perturbations that are supposed to mimic real-world obstacles or create adversarial traffic signs to attack autonomous driving. We focus mainly on recent state-of-the-art optimization-based attacks–CW and EAD–since it has been demonstrated that all earlier attacks can be overcome utilizing other methods, such as adversarial training [15] or defensive distillation [44], which could be used in combination with our approach. Finally, our defense is optimized for edge devices such as smartphones or autonomous vehicles, rather than server or cloud-based classifiers.

IV. Defense Design

Lowering the supply voltage of chips leads to significant power savings. However, reduced voltage operation may also introduce some faults in the device. This is because propagation latency increases at lower supply voltages, which can lead to timing faults. These timing faults can manifest as *bit-flips* in logic outputs or memory. DNNs are known to be resilient to some errors, generally experiencing a graceful degradation in accuracy.

A. Using Undervolting as a Defense Mechanism

It has been shown [11], [19], [28] that adding some amount of random noise to images can help DNNs correctly detect adversarial inputs. The key observation we make in this work is that injecting noise directly in the model improves the ability to detect stronger adversarial inputs, for which correct classification cannot be achieved.

Figure 1 shows the response of a DNN classifier to increasing levels of random noise injected into the model, while classifying both adversarial and benign inputs. We can see that, as the error rate increases the adversarial inputs are less likely to be classified in the intended adversarial class, with their probability of success dropping steeply. Benign inputs are also affected but to a much smaller degree.

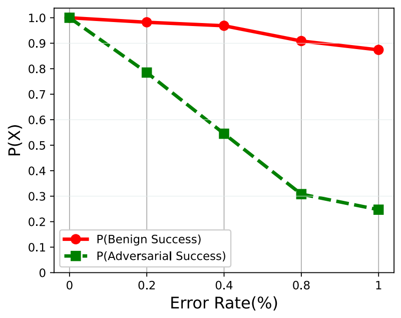

Fig. 1: Probability of success for benign and adversarial classification under the effects of random errors in a DNN model.

We propose using the errors introduced by controlled undervolting of a chip to introduce sufficient perturbations to the model to disrupt adversarial inputs. The proposed system is deployed in three main phases, shown in Figure 2, as follows:

B. Phase I: Offline Device Characterization

In general, the voltage spectrum of a device can be classified into three regions: *Safe, Critical and Crash*. In the *Safe* region, the device operates normally, with no errors. In the *Critical* voltage region the chip will experience occasional errors. Depending on the system and/or application it may continue to operate in the presence of these errors. Finally, the *Crash* voltage is too low for the system to operate. Our defense utilizes voltages in the *Critical* range.

Manufacturing process variation makes the response of each device to undervolting unique. As a result, the critical region resides at different voltage ranges for each chip. The target device therefore needs to be programmed to characterize

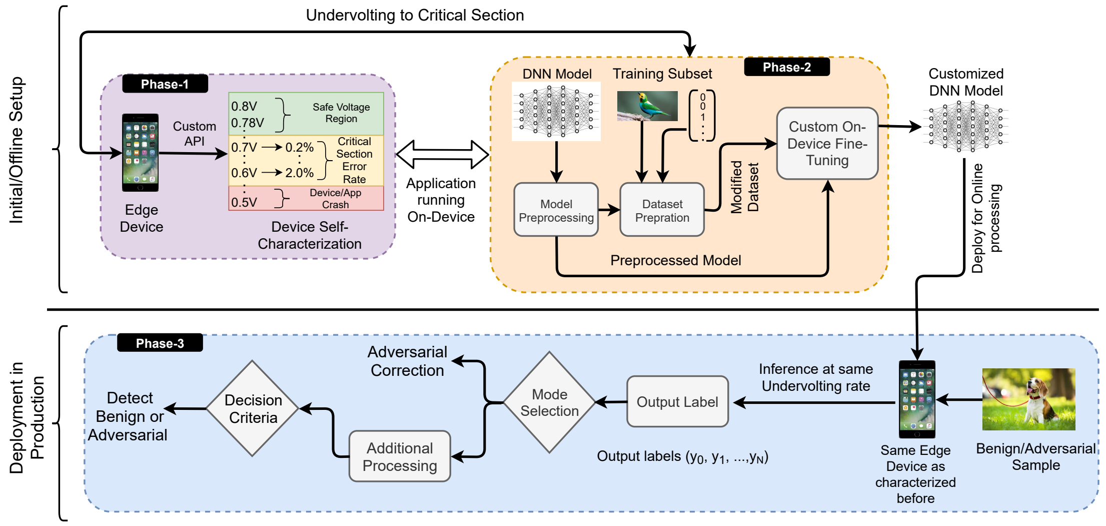

Fig. 2: Overview of proposed defense. A setup phase characterizes the voltage/error profile for each device (Phase 1), and performs on-device model fine-tuning (Phase 2). In production, the device is undervolted and a correction or detection algorithm is applied (Phase 3).

these regions after manufacturing. This characterization can be performed by the device manufacturer, and it involves progressively lowering the supply voltage and running a built-in self test application designed to detect compute or memory errors (Figure 3). Using the error rate profile, the device can undergo controlled undervolting in the critical region during execution.

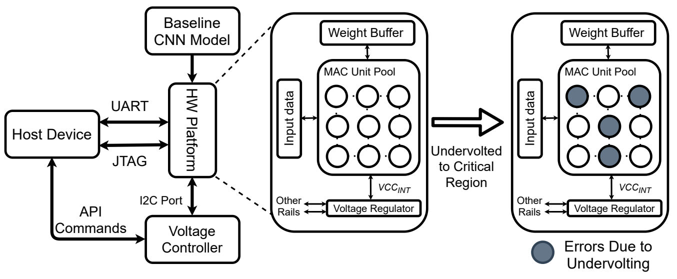

Fig. 3: Characterizing undervolting effects on a hardware DNN accelerator. Error rates are recorded for multiple voltages within the critical region.

C. Phase II: At Setup on-Device Tuning

Our analysis shows that the accuracy of the classification of benign inputs can also be affected by undervolting. To reduce this impact we develop a lightweight on-device fine-tuning process to refine the model to account for errors. This solution takes advantage of the fact that undervolting errors occur in the same functional units of a given chip. In accelerator designs that use mostly static mapping of computation to hardware, this means that errors affect mostly the same model nodes across runs. Fine-tuning the *pre-trained* classifier model in the presence of errors allows the model to adapt and recover most of the accuracy loss. An important constraint on the fine-tuning process is to keep its overhead very low, given that it needs to run on edge devices. This means the fine-tuning should rely on a small training set, to ensure low storage and communication overhead, as well as a small number of training iterations.

Given these constraints, a small subset of less than 0.6% set is sampled randomly from the full training dataset of the model. We explored a range of parameters for the on-device fine tuning, including the number of training epochs and training set size. We choose parameter values at which fine tuning begins to show diminishing returns.

1) Model Preprocessing: Modern CNNs consist of a Softmax layer which is generally used as the last activation function of a neural network to normalize the output vector Z produced by the last hidden layer of a CNN, called *logits*, to a probability distribution over predicted output classes. A single unit of a standard softmax function is denoted as:

$$\sigma_i = \frac{e^{z_i/T}}{\sum\limits_{j=1..N} e^{z_j/T}}$$

where $Z = (z_1, z_2, ..., z_N)$, are the N logit outputs corresponding to N classes in the dataset and T denotes the *temperature* parameter. In general, the T value is set to 1 producing a discrete distribution between 0 and 1. We set T to a higher value in order to increase the uncertainly in the probability distribution vector leading the vector components to converge towards $1/N$ [44]. The softmax layer in the *pre-trained* model is modified with a custom softmax-based activation function. The value of T is model-specific and can be refined along with other parameters such as learning rate and number of epochs, to achieve fast and accurate fine-tuning without overfitting.

2) Dataset Processing: A small training dataset is required for fine-tuning. The dataset selection and processing is highlighted in Step ❶ of Figure 4. A training set consists of (X, y) tuple where $X = (x_0, x_1, .., x_K)$ denoting K input samples/images and $y = (y_{1i}, y_{2i}, .., y_{Ki} | i = 1, 2, .., N)$, denoting N class labels for each of the K inputs. The values for these N class labels are binary, with only the correct class having a value of 1, with the rest 0. The idea of this step is to use the class probability vectors (*soft labels*) produced by the softmax layer instead of the discrete-binary or *hard* class labels to fine-tune the model. Using the probability vectors as the Y values is beneficial as they contain additional information about each class instead of simply providing the correct class label. The intuition is that when a model is being trained, the knowledge is encoded in both the weight parameters and the probability vectors.

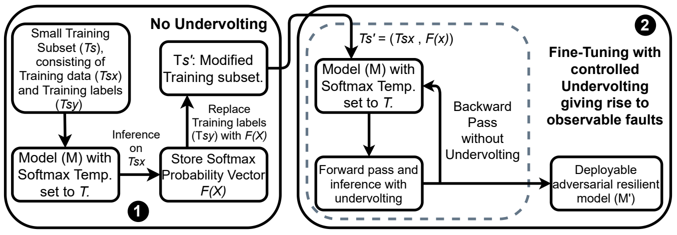
No Undervolting
Small Training Subset (Ts), consisting of Training data (Tsx) and Training labels (Tsy)
Model (M) with Softmax Temp. set to T.
Inference on Tsx
Store Softmax Probability Vector F(X)
Replace Training labels (Tsy) with F(X)
Ts': Modified Training subset.
1
2
Ts' = (Tsx , F(x))
Model (M) with Softmax Temp. set to T.
Forward pass and inference with undervolting
Backward Pass without Undervolting
Fine-Tuning with controlled Undervolting giving rise to observable faults
Deployable adversarial resilient model (M')

Fig. 4: Training subset processing (1) and undervolting-aware model fine-tuning (2).

The second fine-tuning step is ❷ in Figure 4. The targeted chip is undervolted in order to surface the hardware errors and their effect on the model under training. Given that the error distribution does not change significantly over time, the idea is to adjust the model to account for this distribution. The goal is to recover some of the classification accuracy loss caused by the errors to benign inputs. The pre-trained model is fine-tuned with the modified dataset under errors. For a model M with features θ and a given set of sample $(x, P(x) | x \in X)$, where x is the input sample, and $P(x)$ are the soft labels, the goal of this phase is to solve the following optimization problem:

$$\arg\min_{\theta} -\frac{1}{|X|} \sum\nolimits_{x \in X} \sum\nolimits_{i \in N} P_i(x) \log M_i(x)$$

This means for each sample $(x, P(x))$ we consider the log-likelihood $L(M, x, P(x))$ of model M on $(x, P(x))$ and average it for the training set X.

The fine-tuning process requires two passes. The forward pass predicts the output class labels based on some input data. In the backward pass the predicted output is compared with the actual target output. The loss is calculated for that iteration, and is used to update the weights and biases accordingly. The process repeats until it reaches the iterations limit or when the model achieves a threshold accuracy. In our design only the forward pass is undervolted, while the backward pass is performed at nominal voltage to ensure error-free backpropagation of updated parameters. Figure 5 illustrates this process. Once the fine-tuning is complete, the updated model is ready to be deployed in production.

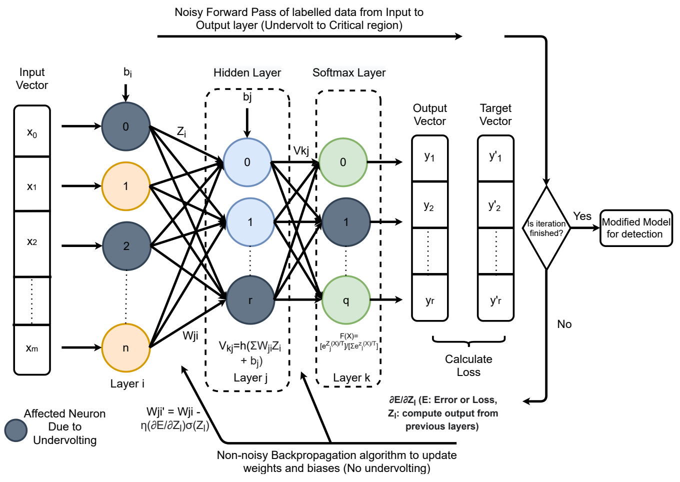
Noisy Forward Pass of labelled data from Input to Output layer (Undervolt to Critical region)
Input Vector
x_0
x_1
x_2
x_m
b_i
0
1
2
n
Layer i
Z_i
Wji
Hidden Layer
bj
0
1
r
$V_{kj}=h(\Sigma W_{ji}Z_i + b_j)$
Layer j
Vkj
Softmax Layer
0
1
q
$F(X)=[e^{Z_j(X)/T}]/[\Sigma e^{Z_l(X)/T}]$
Layer k
Output Vector
y_1
y_2
yr
Target Vector
y'_1
y'_2
y'r
Calculate Loss
Is iteration finished?
Yes
Modified Model for detection
No
$\partial E/\partial Z_l$ (E: Error or Loss, Z_l: compute output from previous layers)
Affected Neuron Due to Undervolting
Wji' = Wji - $\eta(\partial E/\partial Z_l)\sigma(Z_l)$
Non-noisy Backpropagation algorithm to update weights and biases (No undervolting)

Fig. 5: On-device model fine-tuning flow in the presence of undervolting-induced errors.

D. Phase III: Runtime Adversarial Correction/Detection

The final phase of our proposed design consists of the deployment of the modified model in the adversarial defense mechanism. We examine two designs, with different effectiveness and overhead tradeoffs: (1) adversarial correction, which accurately classifies a subset of the adversarial inputs with no performance overhead and lower energy then the baseline, and (2) adversarial detection, which detects >90% of adversarial inputs at the cost of higher performance overhead. We describe the two approaches as follows:

1) Adversarial Correction: The goal of a model or a classifier m, having logit values as k_y, can be given as $F(x) = argmax_y k_y(x)$, where x is an input to the model m. Now, for an adversarial that has been perturbed from the original input x as $(x + \Delta x)$, the goal for an adversarial correction can be summarized as, $F(x + \Delta x) = y = F(x)$, where $y \in (1, ..., K)$, denoting the class labels. By simply running the fine-tuned model on an undervolted chip, the associated errors will induce the classifier to correctly classify a majority of the adversarial inputs. Figure 6 ❶ illustrates the adversarial correction flow. This approach has no performance overhead and, because of undervolting, has lower energy than the unprotected baseline. The correction rate is around 50-60% of the adversarial inputs we test.

2) Adversarial Detection: In order to detect if an input is benign or adversarial, our approach compares the classification output of a reference inference pass at nominal voltage with an undervolted inference pass for the same input. In the first pass, a model with the same characteristics and hyperparameters as the baseline model is selected to get the classification result for the input example *without* undervolting. In the second pass, the modified model is used to get the classification result, *with* undervolting. Both these inference results are compared and, if the results match, the input image is labeled as legitimate. Otherwise, it is flagged as adversarial. Figure 6 ❷ illustrates the adversarial detection process. The intuition behind this

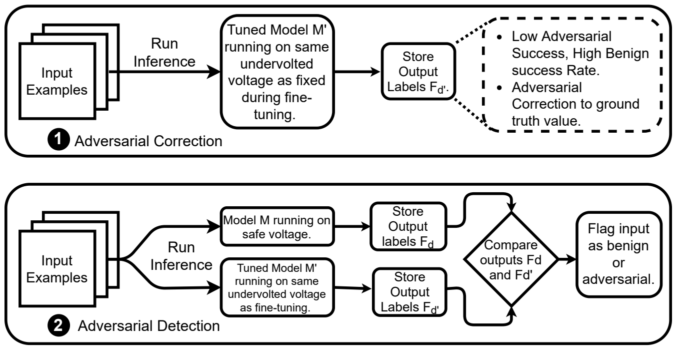
Input Examples
Run Inference
Tuned Model M' running on same undervolted voltage as fixed during fine-tuning.
Store Output Labels F_d'.
Low Adversarial Success, High Benign success Rate.
Adversarial Correction to ground truth value.
1 Adversarial Correction
Input Examples
Run Inference
Model M running on safe voltage.
Tuned Model M' running on same undervolted voltage as fine-tuning.
Store Output labels F_d
Store Output Labels F_d'
Compare outputs Fd and Fd'
Flag input as benign or adversarial.
2 Adversarial Detection

Fig. 6: Runtime adversarial correction and detection mechanisms.

approach is that undervolting is more likely to change the classification (relative to reference run) of the adversarial inputs. Benign inputs are less likely to be affected and their classification will match the reference. We show that this approach is 80-90% effective at detecting adversarial inputs. However, since it requires an error-free reference run, this approach doubles the inference latency – although the energy overhead is only about 70% due to the undervolting power savings.

V. Methodology

A. Evaluation platform

We use CHaiDNN [57], a Deep Neural Network acceleration library designed for Xilinx UltraScale MPSoCs as our evaluation platform. CHaiDNN integrates both system software and a hardware accelerator (Figure 7), supports most of the widely used CNN models, and custom networks and layers.

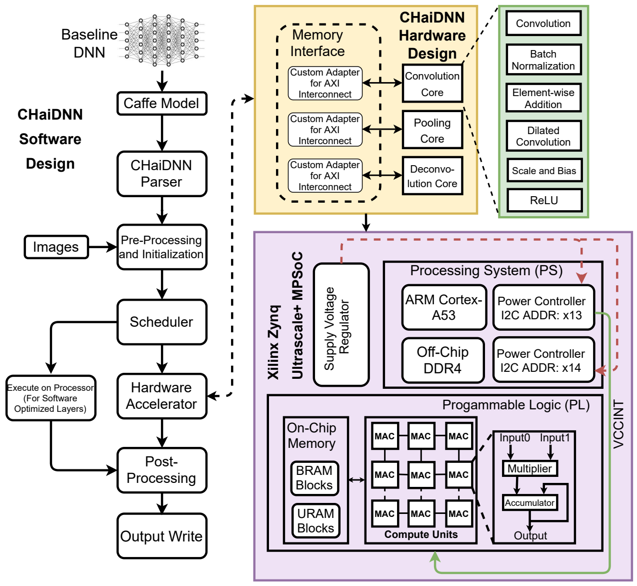

Fig. 7: CHaiDNN accelerator design on ZCU104 platform.

B. FPGA Platform

We deploy CHaiDNN on the Xilinx Zynq Ultrascale+ ZCU104 FPGA platform [2], which uses the XCZU7EV-2FFVC1156 MPSoC. The device includes a quad-core Arm Cortex-A53 applications processor (APU) and a dual-core Cortex-R5 real-time processor (RPU), and a graphics processing unit. The MPSoC is fabricated in a 16nm FinFET+ process technology. The Programming Logic (PL) features about 504K logic blocks (LUTs), 11Mb of Block RAM (BRAM), 27Mb of UltraRAM (URAM) and 1728 signal processing (DSP) slices. The board is also equipped with a 2GB 64-bit DDR4 off-chip memory.

C. Datasets and Classifier Models

We evaluate our defense on two popular convolution-based deep neural networks datasets and models:

1. **CIFAR-10 on DenseNet**: The CIFAR-10 [23] dataset consists of 60,000 32×32 color images in 10 classes, with 6K images per class. There are 50K training images and 10K test images. We use a DenseNet [21], [36] model to evaluate the CIFAR-10 dataset. As Figure 8 shows, DenseNet is composed of an initial convolution layer followed by Dense and Transition blocks and a softmax classifier. In the Dense Blocks, the layers are densely connected.

2. **ImageNet on VGG16**: The ImageNet [25] project is a large visual database designed for use in visual object recognition software research. The most used subset of ImageNet is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [48] image classification and localization dataset. This dataset spans 1K object classes and contains 1,281K training images, 50K validation images and 100K test images. We use the VGG16 [51] model to classify the ImageNet dataset. As Figure 8 shows, VGG16 consists of 16 convolution layers interleaved with activation layers, followed by 3 fully connected layers and a softmax layer for classification.

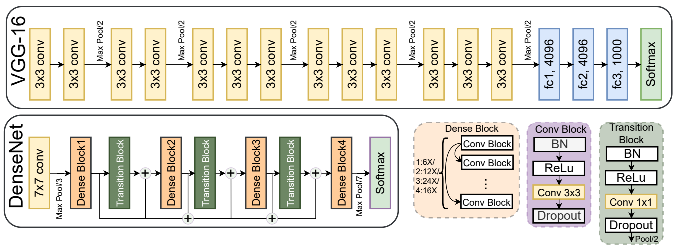

Fig. 8: VGG16 and DenseNet classifier architectures.

D. Attack Types and Datasets

We evaluate our defense on the attacks summarized in Table I. We use two untargeted attacks: FGSM, Deepfool and four targeted attacks: CW_{L0}, EAD_{L1}, CW_{L2}, $CW_{L\inf}$. Targeted attacks aim to misclassify an input into a target class. We deploy two different targets: the *least-likely(LL)* class ($t = min(\hat{y})$) and the *Next* class ($t = L{+}1 \; mod \; \#classes$), where t is the target class, L is the sorted index of top ground truth classes, and $\hat{y}$ is the prediction vector of an input image. We

TABLE I: Evaluated adversarial attack characteristics.

Dataset	Attack	Mode	Success rate	Prediction Conf.	Distortion		
					L_0	L_2	L_∞
$ImageNet$	$FGSM$	-	99%	63.9%	94%	3.1	0.008
	CW_{L0}	LL	100%	84.4%	42.4%	13.65	0.96
		Next	97%	92.9%	42.2%	10.44	0.94
	CW_{L2}	LL	97%	75.25%	54.3%	1.027	0.031
		Next	90%	76.3%	32.2%	0.66	0.019
	$CW_{L\infty}$	LL	100%	91.8%	99.9%	3.05	0.01
		Next	100%	94.7%	100%	2.27	0.01
	EAD_{L1}	LL	100%	100%	54.7%	3.56	0.29
		Next	100%	95.5%	43.8%	3.87	0.7
	$DeepFool$	-	100%	79.59%	98.4%	0.726	0.027
$CIFAR-10$	$FGSM$	-	100%	84.85%	99.7%	0.863	0.016
	CW_{L0}	LL	100%	97.60%	2.4%	2.530	0.712
		Next	100%	98.19%	1.9%	2.103	0.650
	CW_{L2}	LL	100%	97.35%	85.5%	0.358	0.042
		Next	100%	97.90%	76.8%	0.288	0.034
	$CW_{L\infty}$	LL	100%	97.79%	99.5%	0.527	0.014
		Next	100%	98.22%	99.0%	0.446	0.012
	$DeepFool$	-	98%	73.45%	99.5%	0.235	0.028

select a total of 1K legitimate samples and 100 adversarial samples for each attack method from each of the two dataset.

E. Undervolting Methodology

We perform undervolting characterization on the FPGA device using an external voltage controller, the Infineon USB005 [54], which is connected to the board using an I2C cable. The provided PowerIRCenter GUI enables reading and writing the different voltage rails to the board. We can also monitor the power consumption and the temperature of the chip using this GUI. For this study, we focus on VCC_{INT} voltage rail accessible using PMBus address 0x13. This voltage rail drives most of the programming logic components, including the DSPs and LUTs. The layout of our hardware platform setup is illustrated in Figure 9(a).

We gradually sweep the entire voltage range available to our device, while under load, until we reach the crash region. While undervolting in the critical region ($>= 0.660V$ to $< 0.648V$) the CHaiDNN Scheduler application running on the CPU experiences frequent crashes. We attribute this behavior to be the shared voltage rail between the embedded CPU and the Programmable Logic units. In order to better control errors in the Programmable Logic units that contain the DNN accelerator, we inject random errors (as bit flips) directly in the Multiply-Accumulate (MAC) units of the ChaiDNN hardware. The MACs are the dominant components of the accelerator in terms of FPGA resource utilization. Our design deploys 256 such MAC units.

F. Software Simulation

Since the CHaiDNN accelerator does not support training or fine-tuning, we evaluate the error injection and on-device fine-tuning on an Intel Core-i7 CPU@3.40GHz CPU using TensorFlow version 1.14 [1], running the same models as CHaiDNN. We inject errors after every convolution layer of the models which includes activation functions. We also quantize the input and the weights to a fixed INT8 precision and then convolve them, followed by bias addition and activation. The reason for quantizing the inputs and weights is to match the CHaiDNN INT8 computations on the MAC units. Figure 9(b) illustrates the software error model and quantization process.

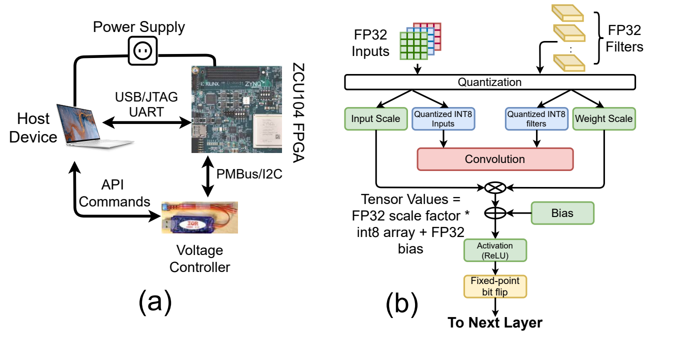

Fig. 9: Hardware platform setup (a) and software model flow (b).

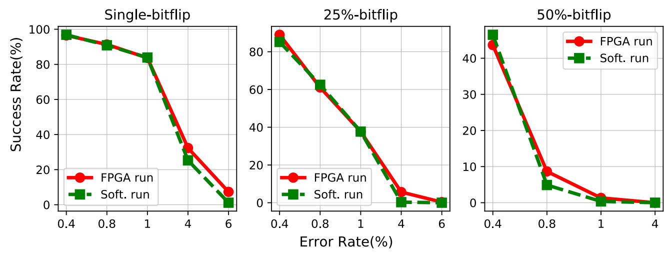

Fig. 10: Undervolting effects on FPGA vs. software model.

This model is next fine-tuned following the process shown in Figure 5. The error distribution is normal and random, with the final results averaged from multiple such distributions. To validate this simulation, we compare the accuracy loss due to undervolting effects on our FPGA platform running the accelerator and the software implementation on a sample benign test set.

Figure 10 shows the effects of undervolting the VGG16 model on the FPGA accelerator compared to the error injection in the software model. We can see that classification accuracy as a function of the error rate matches very well with the FPGA-based injection, across three different error models (single-bit error, 25% bit flips and 50% bit flips). This shows that, with some calibration, we can accurately replicate the effects of FPGA undervolting in the software model. This allows us to evaluate the custom fine-tuning as well as conduct additional sensitivity studies with respect to the error rates.

VI. Evaluation

A. Adversarial Detection and Correction

Figure 11 shows the detection and the correction rates for all adversarial attacks under different simulated error rates for ImageNet and CIFAR-10 examples respectively. Our error model assumes, in this case, that 50% of the bits in each affected output are flipped.

The error rate in the model will vary with the rate of undervolting, affecting the adversarial detection rate, which improves as the rate of error introduced in the model increases. For instance, with a 2% error rate, the average detection rate is at 97% compared to 81% with a 0.02% error rate in the Imagenet dataset. Adversarial correction is evaluated

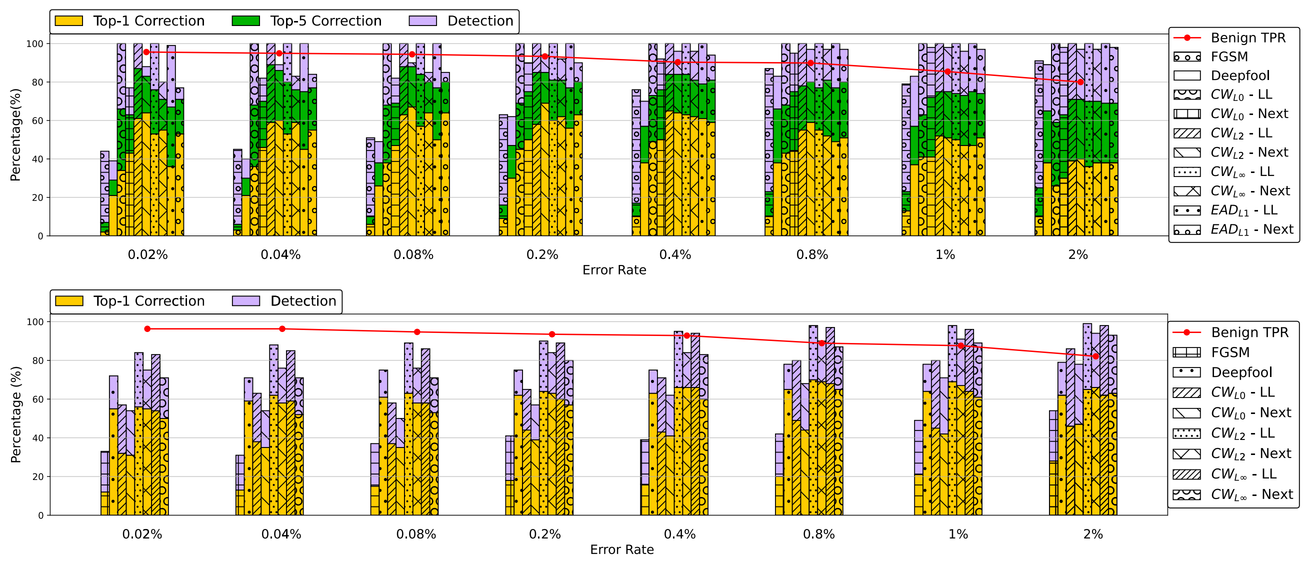

Fig. 11: Adversarial detection/correction rate in ImageNet (top) and CIFAR-10 (bottom). True positive rate (TPR) for benign inputs.

based on the predicted labels of the adversarial examples and comparing the Top-K of these labels with the ground truth of that adversarial. We show Top-1, and Top-5 correction rates for the ImageNet dataset and just Top-1 for CIFAR-10 since the latter only has 10 class labels compared to 1,000 class labels in ImageNet. Figure 11 indicates our defense mechanism is highly effective at reducing the success rate across multiple, diverse adversarial attacks. Our defense works best for low distortion, stronger attacks which are very difficult to detect, such as CW and EAD.

As expected, accuracy for the benign inputs decreases with the increase in error rate. However, the effect of the errors on benign inputs remains modest. The benign *True Positive Rate (TPR)* reaches a low of 80% for 2% error rate. At the same time, it is above 90% for lower error rates at which adversarial detection rates exceed 95%. Fine-grain undervolting allows some control over the error rate. This enables the system to trade-off benign accuracy for adversarial detection rate, to adapt to the needs and constraints of the application.

B. The Benefits of on-Device Fine-Tuning

On-device fine-tuning is designed to mitigate some of the impact of undervolting errors on benign inputs and to improve detection/correction rate through custom distillation. Figure 12 shows the accuracy for benign inputs, and Figure 13 shows the detection and correction rates for adversarial inputs, averaged across all the attacks we study. We evaluate three designs at different error rates:

1) Undervolt: is the core of our design. We evaluate the different models and datasets using just undervolting, with no fine tuning. We study how these errors affect the accuracy of the benign inputs and the success rate for different adversarial inputs. Figure 12 shows that, at low error rates, the accuracy of both models is quite high. However, at higher error rates the accuracy drops to almost 0%. This shows that undervolting alone is only feasible at low error rates, where the adversarial detection rate is less than 80%.

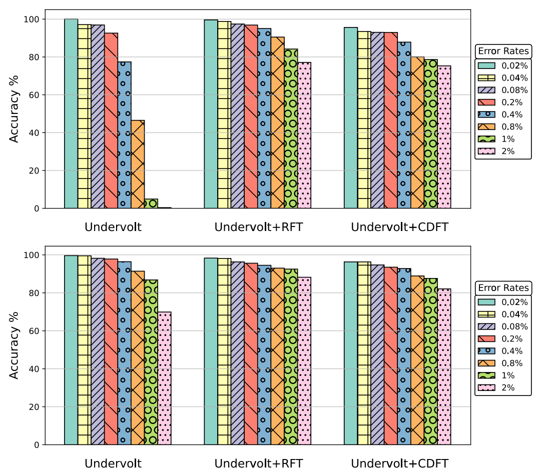

Fig. 12: Fine-tuning impact on benign inputs, ImageNet (top) and CIFAR-10 (bottom).

2) Undervolt + Regular Fine-Tuning (RFT): involves fine-tuning of the model with a small dataset. We observe that fine-tuning improves benign accuracy significantly at high error rates. For the ImageNet dataset, at the highest error rate of 2%, the benign TPR increases from 0% to 80%. We see a 34.4% average increase across all error rates. Fine tuning also improves adversarial correction rates, as Figure 13 shows. The average Top-1 correction rate increases 3.26× for ImageNet and 2.33× for CIFAR-10 vs. undervolting alone.

3) Undervolt + Custom Distillation-Based Fine-Tuning (CDFT): adds a distillation to fine-tuning, as described in Section IV. CDFT is particularly effective at lower error rates and complements undervolting well. For instance, in ImageNet with a 0.02% error rate, we see a 2× increase in adversarial

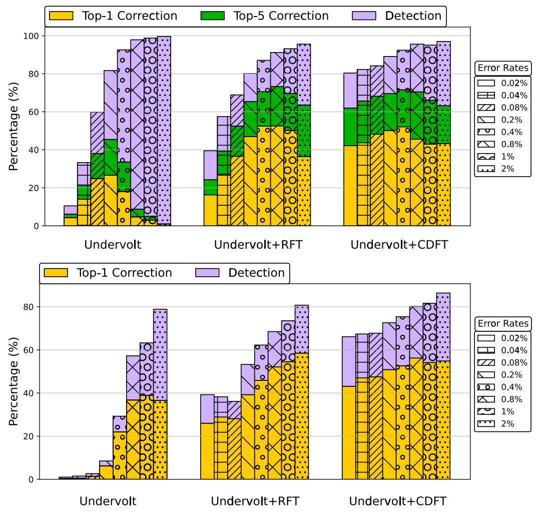

Fig. 13: Fine-tuning impact on adversarial inputs, ImageNet (top) and CIFAR-10 (bottom).

detection rate compared to fine-tuning alone. This is an important result since the adversarial detection/correction rate is significantly lower at low error rates. By adding distillation, we are able to raise the adversarial detection relative to fine-tuning alone, to an average of 90.1% from 76.2% for ImageNet and to 77.34% from 56.2% for CIFAR-10.

C. Sensitivity to Error Profile

We investigate the effect of different error profiles on the effectiveness of our design. We model single bit errors as well as multi-bit errors covering 25% and 50% of the bits in a word. We expect undervolting to produce multi-bit errors, and therefore use 50% bit flips in our main results. For completeness, however, we include results for single bit errors as well as 25% bit errors.

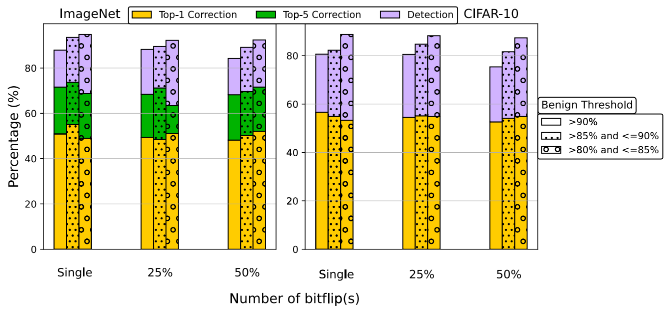

Fig. 14: Adversarial detection and correction for different error profiles at multiple benign true positive rate thresholds.

Figure 14 shows the average detection and correction rates for the three error profiles. We show results for three different thresholds for benign accuracy: >80%, >85% and >90%. For each experiment we use the lowest error rate that is sufficient to ensure the benign accuracy threshold is met. We can see that the correction and detection rates are remarkably consistent across different error profiles for both ImageNet and CIFAR data sets. The detection rates varies by at most 4% between different rates of bit flips and when considering a benign threshold TPR of greater than 90% for both the datasets.

Table II shows a summary of the detection and correction rates across all attacks and error profiles we evaluate for the ImageNet dataset. For each error profile, we use an error rate that is sufficient to achieve a >85% benign TPR for the main defense (Undervolt+CDFT).

D. Comparison with Prior Work

We compare our defense with Feature Squeezing (FS) [58], a defense that manipulates, or ”squeezes”, input images (for example by reducing the color depth and smoothing to reduce the variation among pixels). If the low-resolution image produces substantially different classification outputs than the original image, the input is likely to be adversarial. FS combines multiple squeezers for detection resulting in higher performance overhead than our proposed technique. We applied FS to our set of adversarial images for both ImageNet and CIFAR-10. Figure 15 compares the average adversarial detection rate of FS and our defense for different error rates. We observe that our design compares favorably to FS, even for low error rates. As the error rate increases, our technique performs better, reaching a 97% adversarial detection rate.

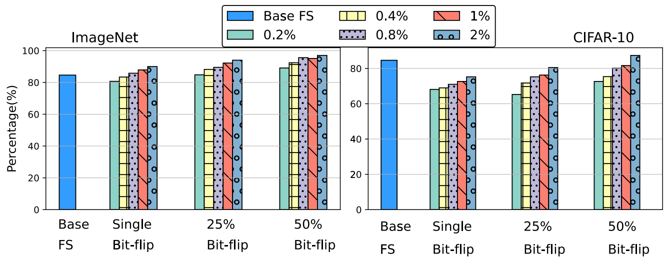

Fig. 15: Adversarial detection compared with Feature Squeezing.

We also observe that FS can complement our defense to achieve higher detection rate. Table-III shows detection rates for a FS variant (using a 2x2 median filter smoothing) compared to our main design (Undervolting+CDFT) and to our design that includes FS (Undervolting+CDFT+FS). We can see that the combined design achieves the highest detection rate for both datasets, demonstrating that undervolting can potentially complement other defenses. However, we do note that the benign TPR decreases slightly for the combined technique. Additional tuning of the combined design might be needed, but was beyond the scope of this work.

E. Resilience to Defense-Aware Attacks

We evaluate our solution against attackers that are aware of the details of our defense. We examine two such attacks:

1) Attack Trained Under Random Noise: In the first attack we assume the adversary is aware of the error-based defense, but has no access to the victim’s hardware (Figure 16 ❶) and does not know the error distribution profile of the victim’s

TABLE II: Detection and correction (Top-1) rate of multiple attacks and variants applied to samples from the ImageNet dataset.

Bit flips	Technique	Benign TPR(%)	L_0 attacks		L_1 attacks		L_2 attacks			L_∞ attacks		
			CW		EAD		Deepfool(%)	CW		FGSM(%)	CW	
			LL(%)	Next(%)	LL(%)	Next(%)		LL(%)	Next(%)		LL(%)	Next(%)
Single	Undervolt	25.3	100/5	100/7	100/5	100/8	98/8	100/6	100/9	87/1	100/4	100/9
	Undervolt+RFT	89.3	100/51	92/51	100/53	88/61	64/37	100/67	96/63	67/8	100/62	96/60
	Undervolt+CDFT	85.42	100/52	93/57	100/62	95/62	79/41	100/66	98/69	79/12	100/66	97/63
25%	Undervolt	62.5	100/12	89/22	98/15	88/26	78/18	100/27	93/30	1/5	100/30	89/19
	Undervolt+RFT	91.2	100/39	82/51	98/45	85/59	59/26	100/57	93/59	48/5	100/64	88/55
	Undervolt+CDFT	87.6	100/43	89/52	100/54	91/59	66/33	100/57	95/60	63/9	100/59	91/58
50%	Undervolt	77.4	100/13	91/19	100/16	93/17	85/17	100/23	93/28	59/4	100/19	97/25
	Undervolt+RFT	93.04	100/48	87/53	100/57	86/58	58/27	100/73	95/69	53/5	100/64	92/60
	Undervolt+CDFT	89.89	100/49	92/51	100/61	94/59	76/38	100/65	96/66	70/10	100/63	96/62

TABLE III: Average adversarial detection rate using a fixed benign TPR threshold for ImageNet/CIFAR-10.

Technique	Benign TPR(%)	Detection(%)
FS(2x2)	93.80/93.20	79.40/85.40
UV+CDFT	92.93/93.50	85.20/81.25
UV+CDFT+FS(2x2)	90.40/91.70	91.60/89.50

device. The attack adds random noise into the model during adversarial construction to attempt to circumvent our defense.

2) Attack Trained on Device: In the second scenario, we assume an attacker with full control of a hardware platform that includes our defense, but is not the device under attack. Control of the device allows the attacker to lean the error distribution of the platform, the model parameters, and can use it indefinitely to create adversarial attacks directly, as shown in Figure 16 ❷. The attack is constructed by training on a device with a pre-determined error profile. This is to determine if an attacker controlling one device can successfully attack a different device, with different error characteristics.

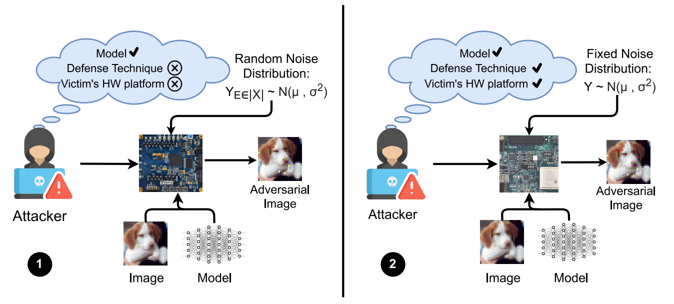

Fig. 16: Defense aware attacks scenarios: (1) trained with random noise and (2) trained on device that includes the defense.

We implement both defense-aware attacks on top of the CW_{L2} attack and use them to generate adversarials based on the ImageNet dataset. All generated adversarials have a baseline success rate of 100%. We re-evaluate these adversarials on a different simulated device with a different error profile. Table IV shows the detection rate for our design for different error rates. We can see that the detection rate for both attacks is very high. This shows that the unique error characteristics of each device ensure a robust defense against defense-aware adversarials.

TABLE IV: Detection rates of defense-aware attacks at different error rates.

Error Rate	Detection Rate	
	Random Noise Attack	Device Trained Attack
0.2%	88%	81%
0.4%	90%	88%
0.8%	96%	95%
1%	99%	98%
2%	100%	100%

F. Energy Impact

Finally, we evaluate the energy impact of the proposed defense, using measurements of power consumption and execution time performed on our FPGA-based implementation of ChaiDNN running VGG16. We measure FPGA power while running the CHaiDNN model, using the external voltage controller. Table V shows inference energy for our defense relative to the unprotected baseline for both correction and detection. Due to the substantial power reduction achieved by undervolting and given that our correction method has no performance overhead, our correction defense achieves about 30% energy savings. Detection requires two sequential inference passes, but due to undervolting this comes at a 70% increase in energy cost instead of $2\times$.

TABLE V: Relative energy for adversarial correction and detection with critical voltage in ZCU104 running CHaiDNN.

Critical Voltage	Relative Energy	
	Correction	Detection
0.660V	0.716	1.716
0.656V	0.714	1.714
0.652V	0.670	1.670

VII. Conclusion

In conclusion, this paper proposes a novel defense against adversarial ML attacks that relies on undervolting . We show that the technique is highly effective, with $>$ 90% detection rate of adversarial inputs for a variety of state-of-the-art attacks. The defense is also energy efficient, due to the power savings from low voltage operation. We also show how the randomness introduced by the unique device characteristics makes our approach robust against defense-aware attacks. We show that the defense is not device specific and can work, with different degrees of effectiveness, depending on the errors produced by undervolting on different devices.

References

[1] "Tensorflow," https://www.tensorflow.org/. [Online]. Available: https://www.tensorflow.org/

[2] "Zynq ultrascale+ mpsoc zcu104 evaluation kit," https://www.xilinx.com/products/boards-and-kits/zcu104.html. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu104.html

[3] A. Bacha and R. Teodorescu, "Dynamic reduction of voltage margins by leveraging on-chip ECC in Itanium II processors," in *International Symposium on Computer Architecture (ISCA)*, June 2013, pp. 297–307.

[4] ——, "Authenticache: Harnessing cache ECC for system authentication," in *International Symposium on Microarchitecture (MICRO)*, December 2015, pp. 1–12.

[5] R. Bertran, A. Buyuktosunoglu, P. Bose, T. J. Slegel, G. Salem, S. Carey, R. F. Rizzolo, and T. Strach, "Voltage noise in multi-core processors: Empirical characterization and optimization opportunities," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 368–380.

[6] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[7] X. Cao and N. Z. Gong, "Mitigating evasion attacks to deep neural networks via region-based classification," in *Proceedings of the 33rd Annual Computer Security Applications Conference on*, 2017, pp. 278–287.

[8] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symposium on Security and Privacy (SP)*, 2017.

[9] K. K. Chang, A. G. Yağlıkçı, S. Ghose, A. Agrawal, N. Chatterjee, A. Kashyap, D. Lee, M. O'Connor, H. Hassan, and O. Mutlu, "Understanding reduced-voltage operation in modern DRAM devices: Experimental characterization, analysis, and mechanisms," in *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, Jun. 2017.

[10] P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, "EAD: elastic-net attacks to deep neural networks via adversarial examples," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[11] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," *arXiv preprint arXiv:1902.02918*, 2019.

[12] G. S. Dhillon, K. Azizzadenesheli, J. D. Bernstein, J. Kossaifi, A. Khanna, Z. C. Lipton, and A. Anandkumar, "Stochastic activation pruning for robust adversarial defense," in *International Conference on Learning Representations (ICLR)*, 2018.

[13] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *International Symposium on Microarchitecture (MICRO)*, December 2003, pp. 7–18.

[14] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1625–1634.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR: International Conference on Learning Representations*, 2015.

[16] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. Mcdaniel, "On the (statistical) detection of adversarial examples," *ArXiv*, vol. abs/1702.06280, 2017.

[17] A. Guesmi, I. Alouani, K. N. Khasawneh, M. Baklouti, T. Frikha, M. Abid, and N. Abu-Ghazaleh, "Defensive approximation: Securing cnns using approximate computing," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. New York, NY, USA: Association for Computing Machinery, 2021, p. 990–1003.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[19] Z. He, A. S. Rakin, and D. Fan, "Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 588–597.

[20] S. Hu, T. Yu, C. Guo, W.-L. Chao, and K. Weinberger, "A new defense against adversarial images: Turning a weakness into a strength," in *NeurIPS 2019 : Thirty-third Conference on Neural Information Processing Systems*, 2019, pp. 1633–1644.

[21] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

[22] M. Kaliorakis, A. Chatzidimitriou, G. Papadimitriou, and D. Gizopoulos, "Statistical analysis of multicore cpus operation in scaled voltage conditions," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 109–112, 2018.

[23] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 1097–1105.

[25] ——, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[26] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *ArXiv*, vol. abs/1611.01236, 2017.

[27] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *International Conference on Learning Representations (ICLR)*, 2017.

[28] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 656–672.

[29] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, and J. B. Carter, "Active management of timing guardband to save energy in POWER7," in *International Symposium on Microarchitecture (MICRO)*, December 2011, pp. 1–11.

[30] J. Leng, Y. Zu, and V. J. Reddi, "GPU voltage noise: Characterization and hierarchical smoothing of spatial and temporal voltage noise interference in gpu architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 161–173.

[31] F. Liao, M. Liang, Y. Dong, T. Pang, J. Zhu, and X. Hu, "Defense against adversarial attacks using high-level representation guided denoiser," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1778–1787, 2018.

[32] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. Van Der Laak, B. Van Ginneken, and C. I. Sánchez, "A survey on deep learning in medical image analysis," *Medical image analysis*, vol. 42, pp. 60–88, 2017.

[33] S. Ma, Y. Liu, G. Tao, W.-C. Lee, and X. Zhang, "NIC: Detecting adversarial samples with neural network invariant checking." in *The Network and Distributed System Security Symposium (NDSS)*, 2019.

[34] X. Ma, B. Li, Y. Wang, S. M. Erfani, S. Wijewickrema, M. E. Houle, G. Schoenebeck, D. Song, and J. Bailey, "Characterizing adversarial subspaces using local intrinsic dimensionality," *arXiv preprint arXiv:1801.02613*, 2018.

[35] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations (ICLR)*, 2018.

[36] S. Majumdar, "Dense Net in Keras," https://github.com/titu1994/DenseNet.

[37] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," *ArXiv*, vol. abs/1702.04267, 2017.

[38] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[39] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 86–94.

[40] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[41] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against Intel SGX," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1466–1482.

[42] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das, “Harnessing voltage margins for energy efficiency in multicore CPUs,” in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 503–516.

[43] N. Papernot, P. Mcdaniel, A. Sinha, and M. P. Wellman, “Towards the science of security and privacy in machine learning,” *ArXiv*, vol. abs/1611.03814, 2016.

[44] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 582–597.

[45] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in *IEEE European Symposium Security and Privacy*, 2016.

[46] M. Pautov, G. Melnikov, E. Kaziakhmedov, K. Kireev, and A. Petiushko, “On adversarial patches: real-world attack on arcface-100 face recognition system,” in *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*. IEEE, 2019, pp. 0391–0396.

[47] R. S. Raju and M. Lipasti, “Blurnet: Defense by filtering the feature maps,” in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2020, pp. 38–46.

[48] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[49] B. Salami, E. B. Onural, I. E. Yuksel, F. Koc, O. Ergin, A. Cristal, O. Unsal, H. Sarbazi-Azad, and O. Mutlu, “An experimental study of reduced-voltage operation in modern FPGAs for neural network acceleration,” *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 138–149, 2020.

[50] B. Salami, O. S. Unsal, and A. Cristal Kestelman, “Comprehensive evaluation of supply voltage underscaling in FPGA on-chip memories,” in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 724–736.

[51] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[52] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, “PixelDefend: Leveraging generative models to understand and defend against adversarial examples,” *ArXiv*, vol. abs/1710.10766, 2018.

[53] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” in *International Conference on Learning Representations*, 2014.

[54] I. Technologies, “Infineon USB005,” https://www.infineon.com/cms/en/product/power/dc-dc-converters/digital-multiphase-controllers/gang-programmers/usb005/.

[55] X. Wang, R. Hou, B. Zhao, F. Yuan, J. Zhang, D. Meng, and X. Qian, “DNNGuard: An elastic heterogeneous dnn accelerator architecture against adversarial attacks,” in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, 2020, p. 19–34.

[56] D. Warde-Farley, “Adversarial perturbations of deep neural networks,” in *Perturbations, Optimization, and Statistics*. MIT Press, 2017, pp. 311–342.

[57] Xilinx, “CHaiDNN,” https://github.com/Xilinx/CHaiDNN.

[58] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks.” in *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.