# Terminal Brain Damage:
# Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks

Sanghyun Hong, *University of Maryland College Park;* Pietro Frigo, *Vrije Universiteit Amsterdam;* Yiğitcan Kaya, *University of Maryland College Park;* Cristiano Giuffrida, *Vrije Universiteit Amsterdam;* Tudor Dumitraş, *University of Maryland College Park*

# Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks

Sanghyun Hong, Pietro Frigo[†], Yiğitcan Kaya, Cristiano Giuffrida[†], Tudor Dumitraș

*University of Maryland, College Park*
[†]*Vrije Universiteit Amsterdam*

## Abstract

Deep neural networks (DNNs) have been shown to tolerate "brain damage": cumulative changes to the network's parameters (e.g., pruning, numerical perturbations) typically result in a graceful degradation of classification accuracy. However, the limits of this natural resilience are not well understood in the presence of small adversarial changes to the DNN parameters' underlying memory representation, such as bit-flips that may be induced by hardware fault attacks. We study the effects of bitwise corruptions on 19 DNN models—six architectures on three image classification tasks—and we show that most models have at least one parameter that, after *a specific bit-flip in their bitwise representation*, causes an accuracy loss of over 90%. For large models, we employ simple heuristics to identify the parameters likely to be vulnerable and estimate that 40–50% of the parameters in a model might lead to an accuracy drop greater than 10% when individually subjected to such single-bit perturbations. To demonstrate how an adversary could take advantage of this vulnerability, we study the impact of an exemplary hardware fault attack, *Rowhammer*, on DNNs. Specifically, we show that a Rowhammer-enabled attacker co-located in the same physical machine can inflict significant accuracy drops (up to 99%) even with single bit-flip corruptions and no knowledge of the model. Our results expose the limits of DNNs' resilience against parameter perturbations induced by real-world fault attacks. We conclude by discussing possible mitigations and future research directions towards fault attack-resilient DNNs.

## 1 Introduction

Deep neural networks (DNNs) are known to be resilient to "brain damage" [32]: typically, cumulative changes to the network's parameters result in a graceful degradation of classification accuracy. This property has been harnessed in a broad range of techniques, such as network pruning [35], which significantly reduces the number of parameters in the network and leads to improved inference times. Besides structural re-

silience, DNN models can tolerate slight noise in their parameters with minimal accuracy degradation [2]. Researchers have proposed utilizing this property in defensive techniques, such as adding Gaussian noise to model parameters to strengthen DNN models against adversarial examples [69]. As a result, this natural resilience is believed to make it difficult for attackers to significantly degrade the overall accuracy by corrupting network parameters.

Recent work has explored the impact of *hardware faults* on DNN models [34, 42, 45]. Such faults can corrupt the memory storing the victim model's parameters, stress-testing DNNs' resilience to bitwise corruptions. For example, Qin et al. [42], confirming speculation from previous studies [34, 35], showed that a DNN model for CIFAR10 image classification does not lose more than 5% accuracy when as many as 2,600 parameters out of 2.5 million are corrupted by *random errors*. However, this analysis is limited to a specific scenario and only considers accidental errors rather than attacker-induced corruptions by means of fault attacks. The widespread usage of DNNs in many mission-critical systems, such as self-driving cars or aviation [12, 51], requires a comprehensive understanding of the security implications of such adversarial bitwise errors.

In this paper, we explore the security properties of DNNs under bitwise errors that can be induced by practical hardware fault attacks. Specifically, we ask the question: *How vulnerable are DNNs to the atomic corruption that a hardware fault attacker can induce?* This paper focuses on single bit-flip attacks that are realistic as they well-approximate the constrained memory corruption primitive of practical hardware fault attacks such as Rowhammer [48]. To answer this question, we conduct a comprehensive study that characterizes the DNN model's responses to single-bit corruptions in each of its parameters.

First, we implement a systematic vulnerability analysis framework that flips each bit in a given model's parameters and measures the misclassification rates on a validation set. Using our framework, we analyze 19 DNN models composed of six different architectures and their variants on three pop-

ular image classification tasks: MNIST, CIFAR10, and ImageNet. Our experiments show that, on average, ~50% of model parameters are vulnerable to single bit-flip corruptions, causing relative accuracy drops above 10%, and that all 19 DNN models include parameters that can cause an accuracy drop of over 90%[1]. The results expose the limits of the DNN's resilience to numerical changes, as adversarial bitwise errors can lead to a *graceless degradation* of classification accuracy.

Our framework also allows us to characterize the vulnerability by examining the impact of various factors: the bit position, bit-flip direction, parameter sign, layer width, activation function, normalization, and model architecture. Our key findings include: 1) the vulnerability is caused by drastic spikes in a parameter value; 2) the spikes in positive parameters are more threatening, however, an activation function that allows negative outputs renders the negative parameters vulnerable as well; 3) the number of vulnerable parameters increases proportionally as the DNN's layers get wider; 4) two common training techniques, e.g., dropout [52] and batch normalization [24], are ineffective in preventing the massive spikes bit-flips cause; and 5) the ratio of vulnerable parameters is almost constant across different architectures (e.g., AlexNet, VGG16, and so on). Further, building on these findings, we propose heuristics for speeding up the analysis of vulnerable parameters in large models.

Second, to understand the practical impact of this vulnerability, we use Rowhammer [26] as an exemplary hardware fault attack. While a variety of hardware fault attacks are documented in literature [11, 26, 38, 57], Rowhammer is particularly amenable to practical, real-world exploitation. Rowhammer takes advantage of a widespread vulnerability in modern DRAM modules and provides an attacker with the ability to trigger controlled memory corruptions directly from unprivileged software execution. As a result, even a constrained Rowhammer-enabled attacker, who only needs to perform a specific memory access pattern, can mount practical attacks in a variety of real-world environments, including cloud [44, 67], browsers [9, 15, 19, 48], mobile [15, 62], and servers [36, 60].

We analyze the feasibility of Rowhammer attacks on DNNs by simulating a Machine-Learning-as-a-Service (MLaaS) scenario, where the victim and attacker VMs are co-located on the same host machine in the cloud. The co-location leads the victim and the attacker to share the same physical memory, enabling the attacker to trigger Rowhammer bit-flips in the victim's data [44, 67]. We focus our analysis to models with an applicable memory footprint, which can realistically be targeted by hardware fault attacks such as Rowhammer.

Our Rowhammer results show that in a *surgical* attack scenario, with the capability of flipping specific bits, the attacker can reliably cause severe accuracy drops in practical settings. Further, even in a *blind* attack scenario, the attacker can still

mount successful attacks without any control over the locations of bit-flips landed in memory. Moreover, we also reveal a potential vulnerability in the *transfer learning* scenario; in which a surgical attack targets the parameters in the layers victim model contains in common with a public one.

Lastly, we discuss directions for viable protection mechanisms, such as reducing the number of vulnerable parameters by preventing significant changes in a parameter value. In particular, this can be done by 1) restricting activation magnitudes and 2) using low-precision numbers for model parameters via quantization or binarization. We show that, when we restrict the activations using the ReLU-6 activation function, the ratio of vulnerable parameters decreases from 47% to 3% in AlexNet, and also, the accuracy drops are largely contained within 10%. Moreover, quantization and binarization reduce the vulnerable parameter ratio from 50% to 1-2% in MNIST. While promising, such solutions cannot deter practical hardware fault attacks in the general case, and often require training the victim model from scratch; hinting that more research is required towards fault attack-resilient DNNs.

**Contributions.** We make three contributions:

- We show DNN models are more vulnerable to bit-flip corruptions than previously assumed. In particular, we show adversarial bitwise corruptions induced by hardware fault attacks can easily inflict severe *indiscriminate damages* by drastically increasing or decreasing the value of a model parameter.

- We conduct the first comprehensive analysis of DNN models' behavior against single bit-flips and characterize the vulnerability that a hardware fault attack can trigger.

- Based on our analysis, we study the impact of practical hardware fault attacks in a representative DL scenario. Our analysis shows that a Rowhammer-enabled attacker can inflict significant accuracy drops (up to 99%) on a victim model even with constrained bit-flip corruptions and no knowledge of the model.

## 2  Preliminaries

Here, we provide an overview of the required background knowledge.

**Deep neural networks.**  A DNN can be conceptualized as a function that takes an input and returns a prediction, i.e., the inferred *label* of the input instance. The network is composed of a sequence of layers that is individually parameterized by a set of matrices, or *weights*. Our work focuses on feed-forward DNNs—specifically on convolutional neural networks (CNNs)—in the supervised learning setting, i.e., the weights that minimize the inference error are learned from

---

[1]The vulnerability of a parameter requires a *specific* bit in its bitwise representation to be flipped. There also might be multiple such bits in the representation that, when flipped separately, trigger the vulnerability.

a labeled *training set*. In a feed-forward network, each layer applies a linear transformation, defined by its weight matrix, to its input—the output of the previous layer—and a bias parameter is added optionally. After the linear transformation, a non-linear *activation function* is applied; as well as other optional layer structures, such as dropout, pooling or batch normalization. During training, the DNN's *parameters*, i.e., the weights in each layer and in other optional structures, are updated iteratively by *backpropagating* the error on the training data. Once the network converges to an acceptable error rate or when it goes through sufficient iterations, training stops and the network, along with all its parameters, is stored as a trained network. During testing (or inference), we load the full model into the memory and produce the prediction for a given input instance, usually not in the training data.

**Single precision floating point numbers.** The parameters of a DNN model are usually represented as IEEE754 32-bit single-precision floating-point numbers. This format leverages the exponential notation and trades off the large range of possible values for reduced precision. For instance, the number 0.15625 in exponential notation is represented as $1.25 \times 2^{-3}$. Here, 1.25 expresses the *mantissa*; whereas $-3$ is the *exponent*. The IEEE754 single-precision floating-point format defines 23 bits to store the mantissa, 8 bits for the exponent, and one bit for the sign of the value. The fact that different bits have different influence on the represented value makes this format interesting from an adversarial perspective. For instance, continuing or example, flipping the 16th bit in the mantissa increases the value from 0.15625 to 0.15625828; hence, a usually negligible perturbation. On the other hand, a flipping the highest exponent bit would turn the value into $1.25 \times 2^{125}$. Although both of these rely on the same bit corruption primitive, they yield vastly different results. In Sec 4, we analyze how this might lead to a vulnerability when a DNN's parameters are corrupted via single bit-flips.

**Rowhammer attacks.** Rowhammer is the most common instance of software-induced fault attacks [9, 15, 19, 44, 48, 60, 62, 67]. This vulnerability provides an aggressor with a single-bit corruption primitive at DRAM level; thus, it is an ideal attack for the purpose of our analysis. Rowhammer is a remarkably versatile fault attack since it only requires an attacker to be able to access content in DRAM; an ubiquitous feature of every modern system. By simply carrying out specific memory access patterns—which we explain in Sec 5—the attacker is able to cause extreme stress on other memory locations triggering faults on other stored data.

## 3   Threat Model

Prior research has extensively validated a DNN's resilience to parameter changes [2, 32, 34, 35, 42, 69], by considering random or deliberate perturbations. However, from a security perspective, these results provide only limited insights as they study a network's expected performance under cumulative changes. In contrast, towards a successful and feasible attack, an adversary is usually interested in inflicting the worst-case damage under minimal changes.

We consider a class of modifications that an adversary, using hardware fault attacks, can induce in practice. We assume a cloud environment where the victim's deep learning system is deployed inside a VM—or a container—to serve the requests of external users. For making test-time inferences, the trained DNN model and its parameters are loaded into the system's (shared) memory and remain constant in normal operation. Recent studies describe this as a typical scenario in MLaaS [61].

To understand the DNNs' vulnerability in this setting, we consider the atomic change that an adversary may induce—the single bit-flip—and we, in Sec 4, systematically characterize the damage such change may cause. We then, in Sec 5, investigate the feasibility of inducing this damage in practice, by considering adversaries with different capabilities and levels of knowledge.

**Capabilities.** We consider an attacker *co-located* in the same physical host machine as the victim's deep learning system. The attacker, due to co-location, can take advantage of a well-known software-induced fault attack, Rowhammer [44,67], for corrupting the victim model stored in DRAM. We take into account two possible scenarios: 1) a *surgical* attack scenario where the attacker can cause a bit-flip at an intended location in the victim's process memory by leveraging advanced memory massaging primitives [44, 62] to obtain more precise results; and 2) a *blind* attack where the attacker lacks fine-grained control over the bit-flips; thus, is completely unaware of where a bit-flip lands in the layout of the model.

**Knowledge.** Using the existing terminology, we consider two levels for the attacker's knowledge of the victim model, e.g., the model's architecture and its parameters as well as their placement in memory: 1) a *black-box* setting where the attacker has no knowledge of the victim model. Here, both the surgical and blind attackers only hope to trigger an accuracy drop as they cannot anticipate what the impact of their bit-flips would be; and 2) a *white-box* setting where the attacker knows the victim model, at least partially. Here, the surgical attacker can deliberately tune the attack's inflicted accuracy drop—from minor to catastrophic damage. Optionally, the attacker can force the victim model to misclassify a specific input sample without significantly damaging the overall accuracy. However, the blind attacker gains no significant advantage over the black-box scenario as the lack of capability prevents the attacker from acting on the knowledge.

# 4 Single-Bit Corruptions on DNNs

In this section, we expose DNNs' vulnerability to single bit-flips. We start with an overview of our experimental setup and methodology. We then present our findings of DNNs' vulnerability to single bit corruptions. For characterizing the vulnerability, we analyze the impact of 1) the bitwise representation of the corrupted parameter, and 2) various DNN properties; on the resulting *indiscriminate damage*[2]. We also discuss the broader implications of the vulnerability for both the blind and surgical attackers. Finally, we turn our attention to two distinct attack scenarios single bit-flips lead to.

## 4.1 Experimental Setup and Methodology

Our vulnerability analysis framework systematically flips the bits in a model, individually, and quantifies the impact using the metrics we define. We implement the framework using Python 3.7[3] and PyTorch 1.0[4] that supports CUDA 9.0 for accelerating computations by using GPUs. Our experiments run on the high performance computing cluster that has 488 nodes, where each is equipped with Intel E5-2680v2 2.8GHz 20-core processors, 180 GB of RAM, and 40 of which have 2 Nvidia Tesla K20m GPUs. We achieve a significant amount of speed-up by leveraging a parameter-level parallelism.

**Datasets.** We use three popular image classification datasets: MNIST [31], CIFAR10 [29], and ImageNet [47]. MNIST is a grayscale image dataset used for handwritten digits (zero to nine) recognition, containing 60,000 training and 10,000 validation images of 28x28 pixels. CIFAR10 and ImageNet are colored image datasets used for object recognition. CIFAR10 includes 32x32 pixels, colored natural images of 10 classes, containing 50,000 training and 10,000 validation images. For ImageNet, we use the ILSVRC-2012 subset [46], resized at 224x224 pixels, composed of 1,281,167 training and 50,000 validation images from 1,000 classes.

**Models.** We conduct our analysis on 19 different DNN models. For MNIST, we define a baseline architecture, Base (B), and generate four variants with different layer configurations: B-Wide, B-PReLU, B-Dropout, and B-DP-Norm. We also examine well-known LeNet5 (L5) [31] and test two variants of it: L5-Dropout and L5-D-Norm. For CIFAR10, we employ the architecture from [55] as a baseline and experiment on its three variants: B-Slim, B-Dropout and B-D-Norm. In the following sections, we discuss why we generate these variants. In Appendix A, we describe the details of these custom architectures; in Appendix C, we present the hyper-parameters. For CIFAR10, we also employ two off-the-shelf

network architectures: AlexNet [30] and VGG16 [50]. For ImageNet, we use five well-known DNNs to understand the vulnerability of large models: AlexNet, VGG16, ResNet50 [22], DenseNet161 [23] and InceptionV3 [56][5].

**Metrics.** To quantify the indiscriminate damage of single bit-flips, we define the Relative Accuracy Drop as $\text{RAD} = (\text{Acc}_{pristine} - \text{Acc}_{corrupted})/\text{Acc}_{pristine}$; where $\text{Acc}_{pristine}$ and $\text{Acc}_{corrupted}$ denote the classification accuracies of the pristine and the corrupted models, respectively. In our experiments, we use [RAD > 0.1] as the criterion for indiscriminate damage on the model. We also measure the accuracy of each class in the validation set to analyze whether a single bit-flip causes a subset of classes to dominate the rest. In MNIST and CIFAR10, we simply compute the Top-1 accuracy on the test data (as a percentage) and use the accuracy for analysis. For ImageNet, we consider both the Top-1 and Top-5 accuracy; however, for the sake of comparability, we report only Top-1 accuracy in Table 1. We consider a parameter as *vulnerable* if it, in its bitwise representation, contains at least one bit that triggers severe indiscriminate damage when flipped. For quantifying the vulnerability of a model, we simply count the number of these vulnerable parameters.

**Methodology.** On our 8 MNIST models, we carry out a complete analysis: we flip each bit in all parameters of a model, in both directions—(0→1) and (1→0)—and compute the RAD over the entire validation. However, a complete analysis of the larger models requires infeasible computational time—the VGG16 model for ImageNet with 138M parameters would take ≈ 942 days on our setup. Therefore, based on our initial results, we devise three speed-up heuristics that aid the analysis of CIFAR10 and ImageNet models.

**Speed-up techniques.** The following three heuristics allow us to feasibly and accurately estimate the vulnerability in larger models:

• *Sampled validation set (SV).* After a bit-flip, deciding whether the bit-flip leads to a vulnerability [RAD > 0.1] requires testing the corrupted model on the validation set; which might be cost prohibitive. This heuristic says that we can still estimate the model accuracy—and the RAD—on a sizable subset of the validation set. Thus, we randomly sample 10% the instances from each class in the respective validation sets, in both CIFAR10 and ImageNet experiments.

• *Inspect only specific bits (SB).* In Sec 2, we showed how flipping different bits of a IEEE754 floating-point number results in vastly different outcomes. Our the initial MNIST analysis in Sec 4.3 shows that mainly the exponent bits lead to perturbations strong enough to cause indiscriminate damage.

---

[2]We use this term to indicate the severe overall accuracy drop in the model.
[3]https://www.python.org
[4]https://pytorch.org

[5]The pre-trained ImageNet models we use are available at: https://pytorch.org/docs/stable/torchvision/models.html

| Dataset | Network | Base acc. | # Params | Speed-up heuristics | | | Vulnerablility | |
|---|---|---|---|---|---|---|---|---|
| | | | | SV | SB | SP | # Params | Ratio |
| MNIST | B(ase) | 95.71 | 21,840 | ✗ | ✗ | ✗ | 10,972 | 50.24% |
| | B-Wide | 98.46 | 85,670 | ✗ | ✗ | ✗ | 42,812 | 49.97% |
| | B-PReLU | 98.13 | 21,843 | ✗ | ✗ | ✗ | 21,663 | 99.18% |
| | B-Dropout | 96.86 | 21,840 | ✗ | ✗ | ✗ | 10,770 | 49.35% |
| | B-DP-Norm | 97.97 | 21,962 | ✗ | ✗ | ✗ | 11,195 | 50.97% |
| | L5 | 98.81 | 61,706 | ✗ | ✗ | ✗ | 28,879 | 46.80% |
| | L5-Dropout | 98.72 | 61,706 | ✗ | ✗ | ✗ | 27,806 | 45.06% |
| | L5-D-Norm | 99.05 | 62,598 | ✗ | ✗ | ✗ | 30,686 | 49.02% |
| CIFAR10 | B(ase) | 83.74 | 776,394 | ✓(83.74) | ✓(exp.) | ✗ | 363,630 | 46.84% |
| | B-Slim | 82.19 | 197,726 | ✓(82.60) | ✓(exp.) | ✗ | 92,058 | 46.68% |
| | B-Dropout | 81.18 | 776,394 | ✓(80.70) | ✓(exp.) | ✗ | 314,745 | 40.54% |
| | B-D-Norm | 80.17 | 777,806 | ✓(80.17) | ✓(exp.) | ✗ | 357,448 | 45.96% |
| | AlexNet | 83.96 | 2,506,570 | ✓(85.00) | ✓(exp.) | ✗ | 1,185,957 | 47.31% |
| | VGG16 | 91.34 | 14,736,727 | ✓(91.34) | ✓(exp.) | ✗ | 6,812,359 | 46.23% |
| ImageNet | AlexNet | 56.52 / 79.07 | 61,100,840 | ✓(51.12 / 75.66) | ✓(31st bit) | ✓(20,000) | 9,467[SP] | 47.34% |
| | VGG16 | 79.52 / 90.38 | 138,357,544 | ✓(64.28 / 86.56) | ✓(31st bit) | ✓(20,000) | 8,414[SP] | 42.07% |
| | ResNet50 | 76.13 / 92.86 | 25,610,152 | ✓(69.76 / 89.86) | ✓(31st bit) | ✓(20,000) | 9,565[SP] | 47.82% |
| | DenseNet161 | 77.13 / 93.56 | 28,900,936 | ✓(72.48 / 90.94) | ✓(31st bit) | ✓(20,000) | 9,790[SP] | 48.95% |
| | InceptionV3 | 69.54 / 88.65 | 27,197,488 | ✓(65.74 / 86.24) | ✓(31st bit) | ✓(20,000) | 8,161[SP] | 40.84% |

SV = Sampled Validation set     SB = Specific Bits     SP = Sampled Parameters set

Table 1: **Indiscriminate damages to 19 DNN models caused by single bit-flips.**

This observation is the basis of our *SB* heuristic that tells us to examine the effects of flipping only the exponent bits for CIFAR10 models. For ImageNet models, we use a stronger SB heuristic and only inspect the most significant exponent bit of a parameter to achieve a greater speed-up. This heuristic causes us to miss the vulnerability the remaining bits might lead to, therefore, its results can be interpreted as a conservative estimate of the actual number of vulnerable parameters.

• *Sampled parameters (SP) set.* Our MNIST analysis also reveals that almost 50% of all parameters are vulnerable to bit-flips. This leads to our third heuristic: uniformly sampling from the parameters of a model would still yield an accurate estimation of the vulnerability. We utilize the SP heuristic for ImageNet models and uniformly sample a fixed number of parameters—20,000—from all parameters in a model. In our experiments, we perform this sampling five times and report the average vulnerability across all runs. Uniform sampling also reflects the fact that a black-box attacker has a uniform probability of corrupting any parameter.

## 4.2 Quantifying the Vulnerability That Leads to Indiscriminate Damage

Table 1 presents the results of our experiments on single-bit corruptions, for 19 different DNN models. We reveal that an attacker, armed with a single bit-flip attack primitive, can successfully cause indiscriminate damage [RAD > 0.1] and that the ratio of vulnerable parameters in a model varies be-

tween 40% to 99%; depending on the model. The consistency between MNIST experiments, in which we examine every possible bit-flip, and the rest, in which we heuristically examine only a subset, shows that, in a DNN model, approximately half of the parameters are vulnerable to single bit-flips. Our experiments also show small variability in the chances of a successful attack—indicated by the ratio of vulnerable parameters. With 40% vulnerable parameters, the InceptionV3 model is the most apparent outlier among the other ImageNet models; compared to 42-49% for the rest. We define the vulnerability based on [RAD > 0.1] and, in Appendix B, we also give how vulnerability changes within the range [0.1 ≤ RAD ≤ 1]. In the following subsections, we characterize the vulnerability in relation to various factors and discuss our results in more detail.

## 4.3 Characterizing the Vulnerability: Bitwise Representation

Here, we characterize the interaction how the features of a parameter's bitwise representation govern its vulnerability.

**Impact of the bit-flip position.** To examine how much change in a parameter's value leads to indiscriminate damage, we focus on the position of the corrupted bits. In Figure 1, for each bit position, we present the number of bits—in the log-scale—that cause indiscriminate damage when flipped, on MNIST-L5 and CIFAR10-AlexNet models. In our MNIST
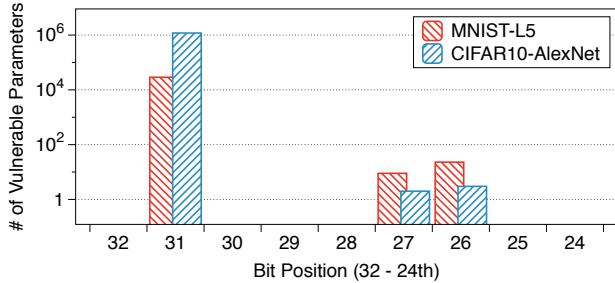
Figure 1: **The impact of the bit position.** The number of vulnerable parameters in bit positions 32nd to 24th.

experiments, we examine all bit positions and we observe that bit positions other than the exponents mostly do not lead to significant damage; therefore, we only consider the exponent bits. We find that *the exponent bits, especially the 31st-bit, lead to indiscriminate damage*. The reason is that a bit-flip in the exponents causes to a drastic change of a parameter value, whereas a flip in the mantissa only increases or decreases the value by a small amount—$[0, 1]$. We also observe that flipping the 30th to 28th bits is mostly inconsequential as these bits, in the IEEE754 representation, are already set to one for most values a DNN parameter usually takes—$[3.0517 \times 10^{-5}, 2]$.

**Impact of the flip direction.** We answer which direction of the bit-flip, ($0 \rightarrow 1$) or ($1 \rightarrow 0$), leads to greater indiscriminate damage. In Table 2, we report the number of effective bit-flips, i.e., those that inflict [RAD > 0.1] for each direction, on 3 MNIST and 2 CIFAR10 models. We observe that *only ($0 \rightarrow 1$) flips cause indiscriminate damage* and *no ($1 \rightarrow 0$) flip leads to vulnerability*. The reason is that a ($1 \rightarrow 0$) flip can only decrease a parameter's value, unlike a ($0 \rightarrow 1$) flip. The values of model parameters are usually normally distributed—$N(0, 1)$—that places most of the values within $[-1,1]$ range. Therefore, a ($1 \rightarrow 0$) flip, in the exponents, can decrease the magnitude of a typical parameter at most by one; which is not a strong enough change to inflict critical damage. Similarly, in the sign bit, both ($0 \rightarrow 1$) and ($1 \rightarrow 0$) flips cannot cause severe damage because they change the magnitude of a parameter at most by two. On the other hand, a ($0 \rightarrow 1$) flip, in the exponents, can increase the parameter value significantly; thus, during the forward-pass, the extreme neuron activation caused by the corrupted parameter overrides the rest of the activations.

| Direction | Models (M: MNIST, C: CIFAR10) | | | | |
|---|---|---|---|---|---|
| (32-24th bits) | **M-B** | **M-PReLU** | **M-L5** | **C-B** | **C-AlexNet** |
| $0 \rightarrow 1$ | 11,019 | 21,711 | 28,902 | 314,768 | 1,185,964 |
| $1 \rightarrow 0$ | 0 | 0 | 0 | 0 | 0 |
| Total | 11,019 | 21,711 | 28,902 | 314,768 | 1,185,964 |

Table 2: **The impact of the flip direction.** The number of effective bit-flips in 3 MNIST and 2 CIFAR10 models.

**Impact of the parameter sign.** As our third feature, we investigate whether the sign—positive or negative—of the corrupted parameter impacts the vulnerability. In Figure 2, we examine the MNIST-L5 model and present the number of vulnerable positive and negative parameters in each layer—in the log-scale. Our results suggest that *positive parameters are more vulnerable to single bit-flips than negative parameters*. We identify the common ReLU activation function as the reason: ReLU immediately zeroes out the negative activation values, which are usually caused by the negative parameters. As a result, the detrimental effects of corrupting a negative parameter fail to propagate further in the model. Moreover, we observe that *in the first and last layers, the negative parameters, as well as the positive ones, are vulnerable*. We hypothesize that, in the first convolutional layer, changes in the parameters yield a similar effect to corrupting the model inputs directly. On the other hand, in their last layers, DNNs usually have the Softmax function that does not have the same zeroing-out effect as ReLU.

## 4.4 Characterizing the Vulnerability: DNN Properties

We continue our analysis by investigating how various properties of a DNN model affect the model's vulnerability to single bit-flips.

**Impact of the layer width.** We start our analysis by asking whether increasing the width of a DNN affects the number of vulnerable parameters. In Table 1, in terms of the number of vulnerable parameters, we compare the MNIST-B model with the MNIST-B-Wide model. In the wide model, all the convolutional and fully-connected layers are twice as wide as the corresponding layer in the base model. We see that the ratio of vulnerable parameters is almost the same for both models: 50.2% vs 50.0%. Further, experiments on the CIFAR10-B-Slim and CIFAR10-B—twice as wide as the slim model—produce consistent results: 46.7% and 46.8%. We conclude that *the number of vulnerable parameters grows proportionally with the DNN's width* and, as a result, *the ratio of vulnerable parameters remains constant at around 50%*.

**Impact of the activation function.** Next, we explore whether the choice of activation function affects the vulnerability. Previously, we showed that ReLU can neutralize the effects of large negative parameters caused by a bit-flip; thus, we experiment on different activation functions that allow negative outputs, e.g., PReLU [21], LeakyReLU, or RReLU [68]. These ReLU variants have been shown to improve the training performance and the accuracy of a DNN. In this experiment, we train the MNIST-B-PReLU model; which is exactly the same as the MNIST-B model, except that it replaces ReLU with PReLU. Figure 3 presents the layer-wise number of
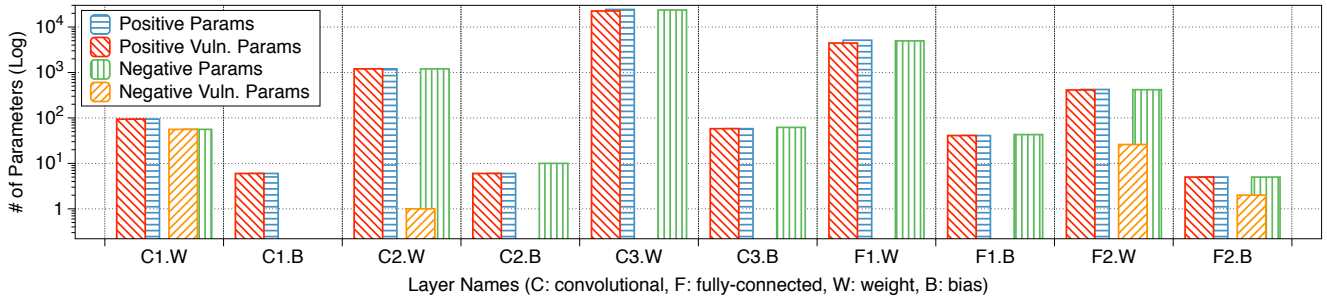
Figure 2: **The impact of the parameter sign.** The number of vulnerable positive and negative parameters, in each layer of MNIST-L5.
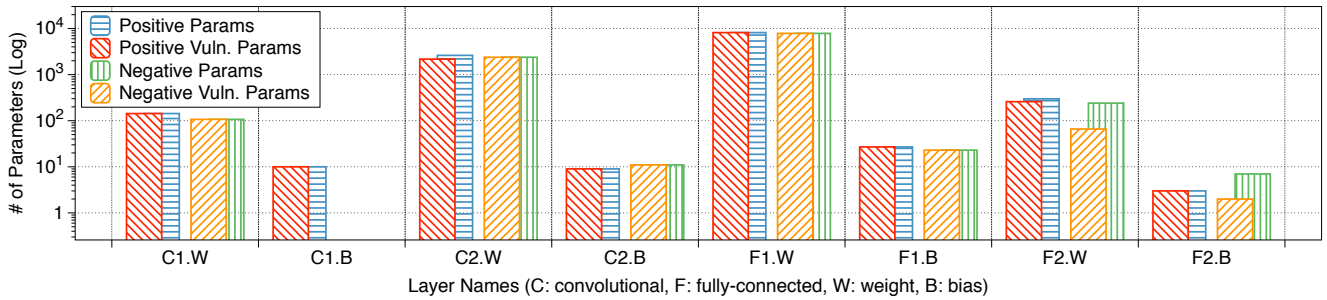


Figure 3: **The impact of the activation function.** The number of vulnerable positive and negative parameters, in each layer of MNIST-PReLU.

vulnerable positive and negative parameters in MNIST-B-PReLU. We observe that *using PReLU causes the negative parameters to become vulnerable and, as a result, leads to a DNN approximately twice as vulnerable as the one that uses ReLU—50.2%* vs. 99.2% vulnerable parameters.
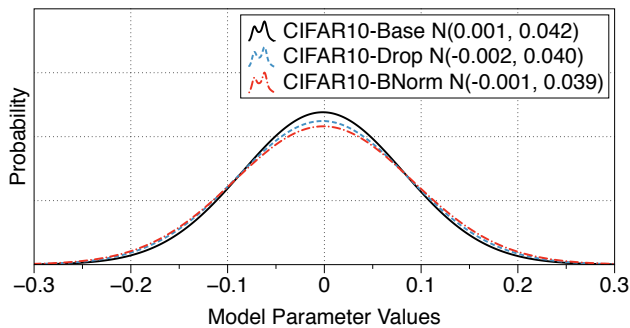


Figure 4: **The impact of the dropout and batch normalization.** The distributions of the parameter values of three CIFAR10 models variants.

**Impact of dropout and batch normalization.** We confirmed that successful bit-flip attacks increase a parameter's value drastically to cause indiscriminate damage. In consequence, we hypothesize that common techniques that tend to constrain the model parameter values to improve the performance, e.g., dropout [52] or batch normalization [24], would

result in a model more resilient to single bit-flips. Besides the base CIFAR10 and MNIST models, we train the B-Dropout and B-DNorm models for comparison. In B-Dropout models, we apply dropout before and after the first fully-connected layers; in B-DNorm models, in addition to dropout, we also apply batch normalization after each convolutional layer. In Figure 4, we compare our three CIFAR10 models and show how dropout and batch normalization have the effect of reducing the parameter values. However, when we look into the vulnerability of these models, we surprisingly find that *the vulnerability is mostly persistent regardless of dropout or batch normalization*—with at most 6.3% reduction in vulnerable parameter ratio over the base network.

**Impact of the model architecture.** Table 1 shows that the vulnerable parameter ratio is mostly consistent across different DNN architectures. However, we see that the InceptionV3 model for ImageNet has a relatively lower ratio—40.8%—compared to the other models—between 42.1% and 48.9%. We hypothesize that the reason is the auxiliary classifiers in the InceptionV3 architecture that have no function at test-time. To confirm our hypothesis, we simply remove the parameters in the auxiliary classifiers; which bring the vulnerability ratio closer to the other models—46.5%. Interestingly, we also observe that the parameters in batch normalization layers are resilient to a bit-flip: corrupting `running_mean` and `running_var` cause negligible damage. In consequence, ex-
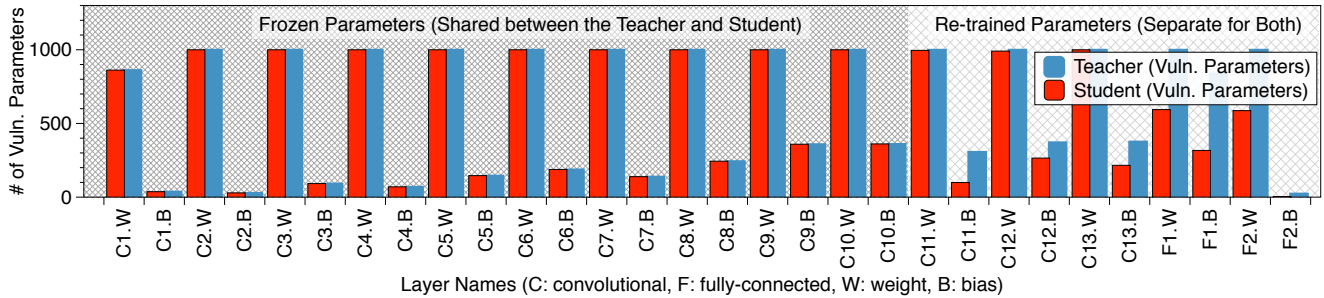
Figure 5: **The security threat in a transfer learning scenario.** The victim model—student—that is trained by transfer learning is vulnerable to the surgical attacker, who can see the parameters the victim has in common with the teacher model.

cluding the parameters in InceptionV3's multiple batch normalization layers leads to a slight increase in vulnerability—by 0.02%.

## 4.5    Implications for the Adversaries

In Sec 3, we defined four attack scenarios: the blind and surgical attackers, in the black-box and white-box settings. First, we consider the strongest attacker: the surgical, who can flip a bit at a specific memory location, white-box, with the model knowledge for anticipating the impact of flipping the said bit. To carry out the attack, this attacker identifies: 1) how much indiscriminate damage, the RAD goal, she intends to inflict, 2) a vulnerable parameter that can lead to the RAD goal, 3) in this parameter, the bit location, e.g., 31st-bit, and the flip direction, e.g., $(0 \rightarrow 1)$, for inflicting the damage. Based on our [RAD > 0.1] criterion, approximately 50% of the parameters are vulnerable in all models; thus, for this goal, the attacker can easily achieve 100% success rate. For more severe goals $[0.1 \leq RAD \leq 0.9]$, our results in Appendix B suggest that the attacker can still find vulnerable parameters. In Sec 5.1, we discuss the necessary primitives, in a practical setting, for this attacker.

For a black-box surgical attacker, on the other hand, the best course of action is to target the 31st-bit of a parameter. This strategy maximizes the attacker's chance of causing indiscriminate damage, even without knowing what, or where, the corrupted parameter is. Considering, the VGG16 model for ImageNet, the attack's success rate is 42.1% as we report in Table 1; which is an upper-bound for the black-box attackers. For the weakest—black-box blind—attacker that cannot specifically target the 31st-bit, we conservatively estimate the lower-bound as 42.1% / 32-bits = 1.32%; assuming only the 31st-bits lead to indiscriminate damage. Note that the success rate for the white-box blind attacker is still 1.32% as acting upon the knowledge of the vulnerable parameters requires an attacker to target specific parameters. In Sec 5.2, we evaluate the practical success rate of a blind attacker.

## 4.6    Distinct Attack Scenarios

In this section, other than causing indiscriminate damage, we discuss two distinct attack scenarios single bit-flips might enable: transfer learning and targeted misclassification.

**Transfer learning scenario.**    Transfer learning is a common technique for *transferring* the knowledge in a pre-trained *teacher* model to a *student* model; which, in many cases, outperforms training a model from scratch. In a practical scenario, a service provider might rely on publicly available teacher as a starting point to train commercial student models. The teacher's knowledge is transferred by *freezing* some of its layers and embedding them into the student model; which, then, trains the remaining layers for its own task. The security risk is that, for an attacker who knows the teacher but not the student, a black-box attack on the student might escalate into a white-box attack on the teacher's frozen layers. The attacker first downloads the pre-trained teacher from the Internet. She then loads the teacher into the memory and waits for the *deduplication* [66] to happen. During deduplication, the memory pages with the same contents—the frozen layers—are merged into the shared pages between the victim and attacker. In consequence, a bit-flip in the attacker's pages can also affect the student model in the victim's memory. We hypothesize that a surgical attacker, who can identify the teacher's vulnerable parameters and trigger bit-flips in these parameters, can cause indiscriminate damage to the student model. In our experiments, we examine two transfer learning tasks in [63]: the traffic sign (GTSRB) [53] and flower recognition (Flower102) [41]. We initialize the student model by transferring first ten frozen layers of the teacher—VGG16 or ResNet50 on ImageNet. We then append a new classification layer and train the resulting student network for its respective task by only updating the new unfrozen layer. We corrupt the 1,000 parameters sampled from each layer in the teacher and monitor the damage to the student model. Figure 5 reports our results: we find that *all vulnerable parameters in the frozen layers and more than a half in the re-trained layers are shared by the teacher and the student.*
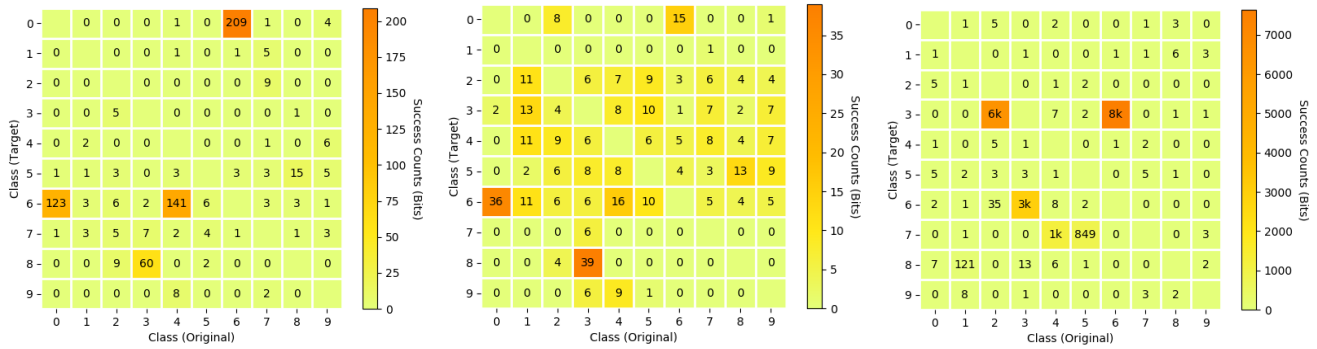
Figure 6: **The vulnerable parameters for a targeted attack in 3 DNN models.** Each cell reports the number of bits that lead to the misclassification of a target sample, whose original class is given by the x-axis, as the target class, which is given by the y-axis. From left to right, the models are MNIST-B, MNIST-L5 and CIFAR10-AlexNet.

**Targeted misclassification.** Although our main focus is showing DNNs' graceless degradation, we conduct an additional experiment to see whether a single bit-flip primitive could be used in the context of *targeted misclassification* attacks. A targeted attack aims to preserve the victim model's overall accuracy while causing the network to misclassify a specific target sample into the target class. We experiment with a target sample from each class in MNIST or CIFAR10—we use MNIST-B, MNIST-L5, and CIFAR10-AlexNet models. Our white-box surgical attacker also preserves the accuracy by limiting the [RAD < 0.05] as in [55]. We find that *the number of vulnerable parameters for targeted misclassifications is lower than that of for causing indiscriminate damage*. In Figure 6, we also see that for some *(original–target class) pairs*, the vulnerability is more evident. For example, in MNIST-B, there are 141 vulnerable parameters for (class 4–class 6) and 209 parameters for (class 6–class 0). Simlarly, in CIFAR10-AlexNet, there are 6,000 parameters for (class 2–class 3); 3,000 parameters for (class 3–class 6); and 8,000 parameters for (class 6–class 3).

## 5 Exploiting Using Rowhammer

In order to corroborate the analysis made in Sec 4 and prove the viability of hardware fault attacks against DNN, we test the resiliency of these models against Rowhammer. At a high level, Rowhammer is a software-induced fault attack that provides the attacker with a single-bit write primitive to specific physical memory locations. That is, an attacker capable of performing specific memory access patterns (at DRAM-level) can induce persistent and repeatable bit corruptions from software. Given that we focus on single-bit perturbations on DNN's parameters in practical settings, Rowhammer represents the perfect candidate for the task.

**DRAM internals.** In Figure 7, we show the internals of a DRAM *bank*. A bank is a bi-dimensional array of memory

*cells* connected to a *row buffer*. Every DRAM chip contains multiple banks. The cells are the actual storage of one's data. They contain a capacitor whose charge determines the value of a specific bit in memory. When a read is issued to a specific row, this row gets *activated*, which means that its content gets transferred to the row buffer before being sent to the CPU. Activation is often requested to recharge a row's capacitors (i.e., *refresh* operation) since they leak charge over time.

**Rowhammer mechanism.** Rowhammer is a DRAM disturbance error that causes spurious bit-flips in DRAM cells generated by frequent activations of a neighboring row. Here, we focus on double-sided Rowhammer, the most common and effective Rowhammer variant used in practical attacks [15, 44, 62]. Figure 8 exemplifies a typical double-sided Rowhammer attack. The victim's data is stored in a row enclosed between two aggressor rows that are repeatedly accessed by the attacker. Due to the continuous activations of the neighboring rows, the victim's data is under intense duress. Thus, there is a large probability of bit-flips on its content.

To implement such attack variant, the attacker usually needs some knowledge or control over the physical memory layout. Depending on the attack scenario, a Rowhammer-enabled attacker can rely on a different set of primitives for this purpose. In our analysis, we consider two possible scenarios: 1) we
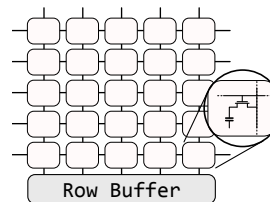


Figure 7: **DRAM bank structure.** Zoom-in on a cell containing the capacitor storing data.
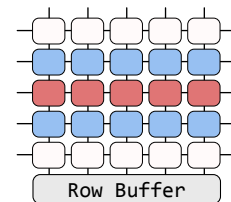
Figure 8: **Double-sided Rowhammer.** Aggressor rows 🟦, and a victim row 🟥 .

| DRAM | # (0→1) flips | DRAM | # (0→1) flips |
|:---:|:---:|:---:|:---:|
| A_2 | 21,538 | A_4 | 5,577 |
| E_2 | 16,320 | I_1 | 4,781 |
| H_1 | 10,608 | J_1 | 4,725 |
| G_1 | 7,851 | E_1 | 4,175 |
| A_1 | 4,367 | A_3 | 1,541 |
| F_1 | 5,927 | C_1 | 1,365 |

Table 3: **Hammertime database [58].** We report the number of (0→1) bit-flips in 12 different DRAM setups. (The rows in gray are used for the experiments in Figure 9.)

initially consider the *surgical* attacker; that is, an attacker with the capability of causing bit-flips at the specific locations, and we demonstrate how, under these assumptions, she can induce indiscriminate damage to a co-located DNN application. 2) We then deprive the attacker of this ability to analyze the outcome of a *blind* attacker and we demonstrate that, even in a more restricted environment, the attacker can still cause indiscriminate damage by causing bitwise corruptions.

**Experimental setup.** For our analysis, we constructed a simulated environment[6] relying on a database of the Rowhammer vulnerability in 12 DRAM chips, provided by Tatar et al [58]. Different memory chips have a different degree of susceptibility to the Rowhammer vulnerability, enabling us to study the impact of Rowhammer attacks on DNNs in different real-world scenarios. Table 3 reports the susceptibility of the different memory chips to Rowhammer. Here, we only include the numbers for (0→1) bit-flips since these are the more interesting ones for the attacker targeting a DNN model according to our earlier analysis in Sec 4.3 and Sec 4.4.

We perform our analysis on an exemplary deep learning application implemented in PyTorch, constantly querying an ImageNet model. We use ImageNet models since we focus on a scenario where the victim has a relevant memory footprint that can be realistically be targeted by hardware fault attacks such as Rowhammer in practical settings. While small models are also potential targets, the number of interesting locations to corrupt is typically limited to draw general conclusions on the practical effectiveness of the attack.

## 5.1 Surgical Attack Using Rowhammer

We start our analysis by discussing a surgical attacker, who has the capability of causing a bit-flip at the specific location in memory. The two surgical attackers are available: the attacker with the knowledge of the victim model (white-

---

| Network | Vuln. Objects (Vuln./Total) | Vuln. Params (in 20k params) | #Hammer Attempts (min/med/max) |
|:---|:---:|:---:|:---:|
| AlexNet | 7/16 | 9,522 | 4/64/4,679 |
| VGG16 | 12/32 | 8,140 | 4/64/4,679 |
| ResNet50 | 9/102 | 3,466 | 4/64/4,679 |
| DenseNet161 | 63/806 | 5,117 | 4/64/4,679 |
| InceptionV3 | 53/483 | 6,711 | 4/64/4,679 |

Table 4: **Effectiveness of surgical attacks.** We examine five different ImageNet models analyzed in Sec 4.

box) and without (black-box). However, in this subsection, we assume that the strongest attacker knows the parameters to compromise and is capable of triggering bit-flips on its corresponding memory location. Then, this attacker can take advantage of accurate memory massing primitives (e.g., memory deduplication) to achieve 100% attack success rate.

**Memory templating.** Since a surgical attacker knows the location of vulnerable parameters, she can *template* the memory up front [44]. That is, the attacker scans the memory by inducing Rowhammer bit-flips in her own allocated chunks and looking for exploitable bit-flips. A surgical attacker aims at specific bit-flips. Hence, while templating the memory, the attacker simplifies the scan by looking for bit-flips located at specific offsets from the start address of a memory *page* (i.e., 4 KB)—the smallest possible chunk allocated from the OS. This allows the attacker to find memory pages vulnerable to Rowhammer bit-flips at a given page offset (i.e., *vulnerable templates*), which they can later use to predictably attack the victim data stored at that location.

**Vulnerable templates.** To locate the parameters of the attacker's interest (i.e., vulnerable parameters) within the memory page, she needs to find *page-aligned* data in the victim model. Modern memory allocators improve performances by storing large objects (usually multiples of the page size) page-aligned whereas smaller objects are not. Thus, we first analyzed the allocations performed by the PyTorch framework running on Python to understand if it performs such optimized page-aligned allocations for large objects similar to other programs [16, 39]. We discovered this to be the case for all the objects larger than 1 MB—i.e., our attacker needs to target the parameters such as weight, bias, and so on, stored as tensor objects in layers, larger than 1 MB.

Then, again focusing on the ImageNet models, we analyzed them to identify the objects that satisfy this condition. Even if the ratio between the total number of objects and target objects may seem often unbalanced in favor of the small ones[7], we found that the number of vulnerable parameters in the target objects is still significant (see Table 4). Furthermore, it is

---

important to note that when considering a surgical attacker, she only needs one single vulnerable template to compromise the victim model, and there is only 1,024 possible offsets where we can store a 4-byte parameter within a 4 KB page.

**Memory massaging.** After finding a vulnerable template, the attacker needs to *massage* the memory to land the victim's data on the vulnerable template. This can be achieved, for instance, by exploiting memory deduplication [9, 44, 67]. Memory deduplication is a system-level memory optimization that merges read-only pages for different processes or VMs when they contain the same data. These pages re-split when a write is issued to them. However, Rowhammer behaves as invisible bit-wise writes that do not trigger the spit, breaking the process boundaries. If the attacker knows (even if partially) the content of the victim model can take advantage of this merging primitive to compromise the victim service.

**Experimental results.** Based on the results of the experiments in Sec 4.3 and Sec 4.4, we analyzed the requirements for a surgical (white-box) attacker to carry out a successful attack. Here, we used one set of the five sampled parameters for each model. In Table 4, we report *min, median, and max* values of the number of rows that an attacker needs to hammer to find the first vulnerable template on the 12 different DRAM setups for each model. This provides a meaningful metric to understand the success rate of a surgical attack. As you can see in Table 4, the results remain unchanged among all the different models. That is, for every model we tested in the best case, it required us to hammer only 4 rows (*A_2* DRAM setup) to find a vulnerable template all the way up to 4,679 in the worst case scenario (*C_1*). The reason why the results are equal among the different models is *due to the number of vulnerable parameters which largely exceeds the number of possible offsets within a page that can store such parameters* (i.e., 1024). Since every vulnerable parameter yields indiscriminate damage [RAD > 0.1], we simply need to identify a template that could match any given vulnerable parameter. This means that an attacker can find a vulnerable template at best in a matter of few seconds[8] and at worst still within minutes. Once the vulnerable template is found, the attacker can leverage memory deduplication to mount an effective attack against the DNN model—with no interference with the rest of the system.

## 5.2 Blind Attack Using Rowhammer

While in Sec 5.1 we analyzed the outcome of a surgical attack, here we abstract some of the assumptions made above and study the effectiveness of a blind attacker oblivious of the bit-flip location in memory. To bound the time of the lengthy

blind Rowhammer attack analysis, we specifically focus our experiments on the ImageNet-VGG16 model.

We run our PyTorch application under the pressure of Rowhammer bit-flips indiscriminately targeting both code and data regions of the process's memory. Our goal is twofold: 1) to understand the effectiveness of such attack vector in a less controlled environment and 2) to examine the robustness of a running DNN application to Rowhammer bit-flips by measuring the number of failures (i.e., crashes) that our blind attacker may inadvertently induce.

**Attacker's capabilities.** We consider a blind attacker who cannot control the bit-flips caused by Rowhammer. As a result, the attacker may corrupt bits in the DNN's parameters as well as the code blocks in the victim process's memory. In principle, since Rowhammer bit-flips propagate at the DRAM level, a fully blind Rowhammer attacker may also inadvertently flip bits in other system memory locations. In practice, even an attacker with limited knowledge of the system memory allocator, can heavily influence the physical memory layout by means of specially crafted memory allocations [17, 18]. Since this strategy allows attackers to achieve co-location with the victim memory and avoid unnecessary fault propagation in practical settings, we restrict our analysis to a scenario where bit-flips can only (blindly) corrupt memory of the victim deep learning process. This also generalizes our analysis to arbitrary deployment scenarios, since the effectiveness of blind attacks targeting arbitrary system memory is inherently environment-specific.

**Methods.** For every one of the 12 vulnerable DRAM setups available in the database, we carried out 25 experiments where we performed at most 300 "hammering" attempts—value chosen after the surgical attack analysis where a median of 64 attempts was required. The experiment has three possible outcomes: 1) we trigger one(or more) effective bit-flip(*s*) that compromise the model, and we record the relative accuracy drop when performing our testing queries; 2) we trigger one(or more) effective bit-flip(*s*) in other victim memory locations that result in a crash of the deep learning process; 3) we reach the "timeout" value of 300 hammering attempts. We set such "timeout" value to bound our experimental analysis which would otherwise result too lengthy.

**Experimental results.** In Figure 9, we present the results for three sampled DRAM setups. We picked *A_2*, *I_1*, and *C_1* as representative samples since they are the most, least, and moderately vulnerable DRAM chips (see Table 3). Depending on the DRAM setup, we obtain fairly different results. We found *A_2* obtains successful indiscriminate damages to the model in 24 out of 25 experiments while, in less vulnerable environments such as *C_1*, the number of successes decreases to only one while the other 24 times out. However, it is im-

---
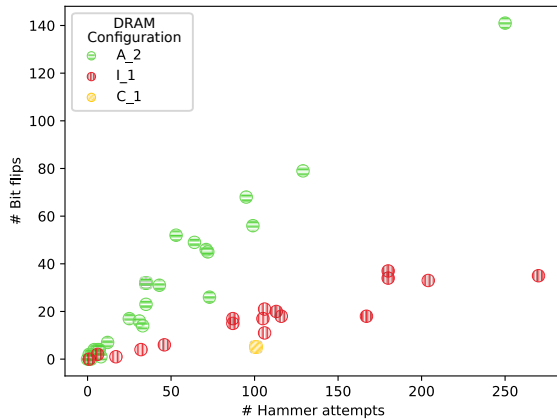
[8]We assume 200ms to hammer a row.

Figure 9: **The successful runs of a blind attack execution over three different DRAM setups (*A_2*-most, *I_1*-least, and *C_1*-moderately vulnerable).** We report the success in terms of #*flips* and #*hammer attempts* required to obtain an indiscriminate damage to the victim model. We observe the successes within few hammering attempts.
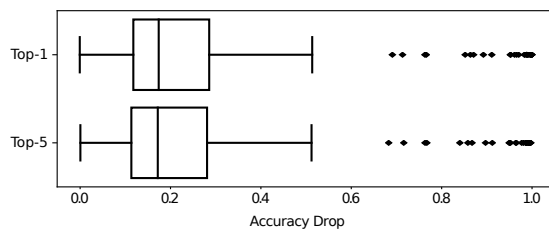


Figure 10: **The distribution of relative accuracy drop for Top-1 and Top-5.** We compute them over the effective #*flips* in our experiments on the ImageNet-VGG16 model.

portant to note that a timeout does not represent a negative result—a crash. Contrarily, while *C_1* only had a single successful attack, it also represents a peculiar case corroborating the analysis presented in Sec 4. The corruption generated in this single successful experiment was induced by a single bit-flip, which caused one of the most significant RADs detected in the entire experiment, i.e., 0.9992 and 0.9959 in Top-1 and Top-5. Regardless of this edge case, we report a mean of 15.6 out of 25 effective attacks for this Rowhammer variant over the different DRAM setups. Moreover, we report the distribution of accuracy drops for Top-1 and Top-5 in Figure 10. In particular, the median drop for Top-1 and Top-5 confirms the claims made in the previous sections, i.e., the blind attacker can expect [RAD > 0.1] on average.

Interestingly, when studying the robustness of the victim process to Rowhammer, we discovered it to be quite resilient to spurious bit-flips. We registered only 6 crashes over all the different DRAM configurations and experiments—300

in total. This shows that the model effectively dominates the memory footprint of the victim process and confirms findings from our earlier analysis that bit-flips in non-vulnerable model elements have essentially no noticeable impact.

## 5.3 Synopsis

Throughout the section, we analyzed the outcome of surgical and blind attacks against large DNN models and demonstrated how Rowhammer can be deployed as a feasible attack vector against these models. These results corroborate our findings in Sec 4 where we estimated at least 40% of a model's parameters to be vulnerable to single-bit corruptions. Due to this large attack surface, in Sec 5.1, we showed that a Rowhammer-enabled attacker armed with knowledge of the network's parameters and powerful memory massaging primitives [44, 62, 67] can carry out precise and effective indiscriminate attacks in a matter of, at most, few minutes in our simulated environment. Furthermore, this property, combined with the resiliency to spurious bit-flips of the (perhaps idle) code regions, allowed us to build successful blind attacks against the ImageNet-VGG16 model and inflict "terminal brain damage" even when hiding the model from the attacker.

## 6 Discussion

In this section, we discuss and evaluate some potential mitigations to protect against single-bit attacks on DNN models. We discuss two research directions towards making DNN models resilient to bit-flips: *to restrict activation magnitudes* and *to use low-precision numbers*. Prior work on defenses against Rowhammer attacks suggest system-level defenses [10, 27] that often even require specific hardware support [6, 26]. Yet they have not been widely deployed since they require infrastructure-wide changes from cloud host providers. Moreover, even though the infrastructure is robust to Rowhammer attacks, an adversary can leverage other vectors to exploit bit-flips attacks to corrupt a model. Thus, we focus on the solutions that our victim can apply to his models.

## 6.1 Restricting Activation Magnitudes

In Sec 4.3, we found that the vulnerable parameter ratio changes based on inherent properties of a DNN; for instance, using PReLU activation function allows a model to propagate negative extreme activations. Hence, if we opt for an activation function that always bounds the output within a specific range, a bit-flip is hard to cause indiscriminate damage. There are several functions, such as Tanh or HardTanh [25], that suppresses the activations; however, using ReLU-6 [28] function provides two key advantages over the others: 1) the victim only needs to substitute the existing activation functions from ReLU to ReLU-6 without re-training, and 2) ReLU-6 allows

| Network | Train | Base acc. | # Params | Vulnerability |
|---|---|---|---|---|
| Base (ReLU) | Scr | 98.13 | | 10,972 (50.2%) |
| Base (ReLU6) | Scr | 98.16 | 21,840 | 313 (1.4%) |
| Base (Tanh) | Scr | 97.25 | | 507 (2.3%) |
| Base (ReLU6) | Sub | 95.71 | | 542 (2.4%) |
| AlexNet (ReLU) | - | 56.52 / 79.07 | | 9,467 (47.34%) |
| AlexNet (ReLU6) | Sub | 39.80 / 65.82 | 20,000 (61M) | 560 (2.8%) |
| AlexNet (ReLUA) | Sub | 56.52 / 79.07 | | 1,063 (5.32%) |
| VGG16 (ReLU) | - | 64.28 / 86.56 | | 8,227 (41.13%) |
| VGG16 (ReLU6) | Sub | 38.58 / 64.84 | 20,000 (138M) | 2,339 (11.67%) |
| VGG16 (ReLUA) | Sub | 64.28 / 86.56 | | 2,427 (12.14%) |

Table 5: **Effectiveness of restricting activation.**

the victim to control the level of permitted activation by modifying the bounds, e.g., using other limits instead of 6, which minimizes the performance loss by bounding the activation. What the victim can do is to monitor the activation values over the validation set and to decide the limits that only suppresses the abnormal activation by bit-flips. For example, in our experiments with ImageNet-AlexNet, we set the limits to $[0, max]$, where $max$ is defined adaptively by looking at the maximum activation from each layer (ReLU-A).
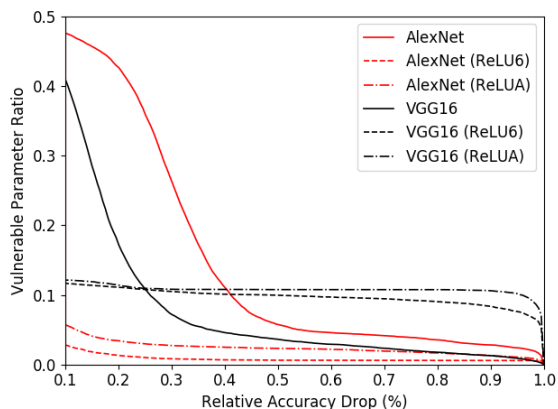


Figure 11: **The vulnerability of DNN models using different criterions.** We illustrate the ImageNet-AlexNet (red lines) and -VGG16 (black lines) cases with ReLU-6 and ReLU-A.

**Experiments.** We use three DNN models in Sec 4: the MNIST-B, ImageNet-AlexNet, and ImageNet-VGG16 models. We evaluate four activation functions: ReLU (default), Tanh, ReLU-6, and ReLU-A (only for AlexNet and VGG16), and two training methods: training a model from scratch (Scr) or substituting the existing activation into another (Sub). We use the notation as the network names with the activation in parenthesis, e.g., AlexNet (ReLU-6). For larger models, we use the same speed-up heuristics in Sec 4.2: SV, SB, and SP.

Table 5 shows the effectiveness of our proposal. For each network (Column 1), we list the training method, the base

| Network | Method | Base acc. | # Params | Vulnerability |
|---|---|---|---|---|
| L5 | - | 99.24 | 62,598 | 30,686 (49.0%) |
| L5 | 8-bit Quantized | 99.03 | 62,600 | 0 (0.0%) |
| L5 | XNOR Binarized | 98.39 | 62,286 | 623 (1.0%) |

Table 6: **Effectiveness of using low-precision.**

accuracy, the number of examined parameters, and the vulnerability (Column 2-5). We found that restricting activation magnitudes with Tanh and ReLU-6 in some instances can reduce the vulnerability; For instance, in the MNIST models, we observed that the number of vulnerable parameters is reduced from 50% to 1.4-2.4% without incurring in significant performance loss. Further, we discovered that ReLU-6 achieves a similar effect without re-training of a model like Tanh. However, there are the vulnerable parameters after the restrictions since we cannot apply the ReLU-6 function to the last layer. In AlexNet and VGG16, the decrease in the number of vulnerable parameters is also generally significant, namely from 47.34% to 2.8% and 41.13% to 11.67%. However, we observe the models suffer from large accuracy drops caused by restricting the activation. To minimize the loss, we control the bounds of activation in AlexNet (ReLU-A) and VGG16 (ReLU-A) by choosing the maximum activation from each layer. With the ReLU-A, we can trade accuracy for the number of vulnerable parameters as we show in Table 5. Nevertheless, it is interesting to see that by employing ReLU-A, while the number of vulnerable parameters remains significant, the RAD also suffers from the new activation function limiting the possible effects of the corruption. In Figure 11, the dashed lines are for ReLU-6, the dashed-dot lines are for ReLU-A, and the straight lines are for ReLU. We found the ReLU-A lines are between the ReLU and ReLU-6 in AlexNet.

**Takeaways.** Our experimental results with restricting activation magnitudes suggest that: this mechanism 1) allows a defender to control the trade-off between the relative accuracy drop and reducing the vulnerable parameters and 2) enables ad-hoc defenses to DNN models, which does not require training the network from scratch. However, the remaining number of vulnerable parameters shows that the Rowhammer attacker still could inflict damage, with a reduced success rate.

### 6.2 Using Low-precision Numbers

Another direction is to represent the model parameters as low-precision numbers by using quantization and binarization. In Sec 4.3, we found that the vulnerability exploits the bit-wise representation of the corrupted parameter to induce the dramatic chances in the parameter value. Our intuition is to use low-precision numbers hard to be increased dramatically by a bit-flip; for example, an integer expressed as the 8-bit quantized format can be increased at most 128 by a flip in the

MSB (8th bit). Thus, the attacker only can increase a model parameter with such a restricted bound. Training models using low-precision numbers are supported by the popular deep learning frameworks such as TensorFlow[9]. The victim can train and deploy the model with the quantized or binarized parameters by utilizing the frameworks.

**Experiments.** To validate our intuition, we use 3 DNN models: the MNIST-L5 (baseline) and its quantized and binarized models. When we quantize the MNIST-L5 model, we use the 8-bit quantization in [7, 64] which converts the model parameters in all layers into integers between 0 and 255. For the binarization, we employ the method in XNOR-Net [43] which converts the model parameters to -1 and 1, except the first convolutional layer. Using the trained models, we evaluate the vulnerability to single bit-flips, and report the accuracy, total parameters, and vulnerability, without the speed-up heuristics.

Table 6 shows the effectiveness of using low-precision parameters. For each network (Column 1), we note the quantization method, the accuracy, the number of vulnerable parameters, and their percentage (Column 2-5). We found that *using low-precision parameters reduces the vulnerability*; in all cases, the percentage of vulnerable parameters are reduced from 49% (Baseline) to 0-2% (surprisingly 0% with the quantization). We focus on analyzing which layer has vulnerable parameters in the binarization model. We found that mostly the parameters in the first convolutional (150 parameters) and classification (last) layers (420 parameters) are vulnerable to a bit-flip, which corroborates what observed in Sec 4.3.

**Takeaways.** Even though we showed the elimination of the vulnerability through 8-bit quantization, in a real-world, training the large model such as [65] from scratch can take a week on a supercomputing cluster.

# 7 Related Work

**DNN's resilience to perturbations.** Prior work has utilized the *graceful degradation* of DNN models under parameter perturbations in a wide range of applications. For example, network quantization [3, 5], by quantizing a DNN model's high-precision parameter into low-precision, reduces the size and inference time of a model with negligible performance penalty. This property has also been used as a primitive for improving the security of DNNs. For example, modifying the parameter slightly to inject a watermark to allow model owners to prove ownership [1]; adding Gaussian noise to model parameter for reducing the reliability of test-time adversarial attacks on DNNs [69]; and fine-tuning the parameters for mitigating the malicious backdoors in a model [37]. Further,

the resilience to structural changes has lead to pruning techniques [4, 20, 35] which improve the efficiency of a DNN model by removing unimportant neurons along with their parameters. In our work, we study the *graceless degredation* of DNNs under hardware fault attacks that induce single bit-flips to individual parameters.

**Indiscriminate poisoning attacks on DNNs.** Recent work on adversarial machine learning has demonstrated many attack scenarios to inflict indiscriminate damage on a model. One of the well-studied vectors is *indiscriminate poisoning attacks* [8] in which the adversary, by injecting malicious data in the victim's training set, aims to hurt the model. Previous studies suggest that such attack might require significant amount of poisonous instances [40]. For example, Steinhardt et al. [54] shows that, with IMDB dataset, an attacker needs to craft 3% of the total training instances to achieve 11% of accuracy drop compared to the pristine model. Further, the defenses based on robust outlier removal techniques could render poison injection ineffective by filtering it out [14, 54]. Moreover, to achieve targeted damages without harming the model's overall accuracy, *targeted poisoning attacks* [49, 55] have been studied. In this paper, we analyze a test-time vulnerability that does not require the adversary's contact to the victim model during its training. This vulnerability inflicts indiscriminate damage, similar to indiscriminate poisoning attacks, through a different attack medium.

**Hardware fault injection attacks.** Hardware fault injection is a class of attacks that rely on hardware glitches on the system to corrupt victim's data. These glitches generally provide a single-bit write primitive at the physical memory; which could potentially lead to privilege escalation [67]. While in the past these attacks required physical access to the victim's system [11, 38], recently they have gained more momentum since the software-based version of these attacks were demonstrated [26, 57]. Instances of these attacks are 1) the CLKSCREW attack [57] that leverages dynamic voltage and frequency scaling on mobile processors generate faults on instructions; or 2) the well-known Rowhammer vulnerability that triggers bitwise corruptions in DRAM. Rowhammer has been used in the context of cloud VMs [44, 67], on desktops [48] and mobile [62] and even to compromise browsers from JavaScript [9, 15, 19]. In the context of DNNs, fault attacks have been proposed as an alternative for inflicting indiscriminate damages. Instead of injecting poisonous instances, fault attacks directly induce perturbations to the models running on hardware [11, 13, 34, 38, 45]. These studies have considered the adversaries with direct access to the victim hardware [11, 13] and adversaries who randomly corrupt parameters [34, 38, 45]. We utilize Rowhammer as an established fault attack to demonstrate practical implications of the graceless degradation of DNNs. Our threat model follows the realistic single bit-flip capability of a fault attack and modern

---

[9] https://www.tensorflow.org/lite/performance/post_training_quantization

application of DNNs in a cloud environment, where physical access to the hardware is impractical.

## 8    Conclusions

This work exposes the limits of DNN's resilience against the parameter perturbations. We study the vulnerability of DNN models to single bit-flips. We evaluated 19 DNN models with six architectures on three image classification tasks and showed that: we can easily find 40-50% vulnerable parameters where an attacker can cause indiscriminate damage [RAD > 0.1] by a bit-flip. We further characterize this vulnerability based on the impact of various factors: the bit position, bit-flip direction, parameter sign, layer width, activation function, training techniques, and model architecture. Understanding this emerging threat, we leverage the software-induced fault injection, Rowhammer, to demonstrate the feasibility of the bit-flip attacks in practice. In experiments with RowHammer, we found that, without knowing the victim's deep learning system, the attacker can inflict indiscriminate damage without system crashes. Lastly, motivated by the attacks, we discuss two potential directions of mitigation: restricting activation magnitudes and using low-precision numbers.

## Acknowledgments

## References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1615–1631, Baltimore, MD, 2018. USENIX Association.

[2] G. An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3):643–674, April 1996.

[3] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1131–1135. IEEE, 2015.

[4] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017.

[5] Yiwen Guo Lin Xu Yurong Chen Aojun Zhou, Anbang Yao. Incremental network quantization: Towards lossless cnns with low-precision weights. In *International Conference on Learning Representations (ICLR)*, 2017.

[6] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. Anvil: Software-based protection against next-generation rowhammer attacks. *ACM SIGPLAN Notices*, 51(4):743–755, 2016.

[7] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5145–5153. Curran Associates, Inc., 2018.

[8] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1467–1474, USA, 2012. Omnipress.

[9] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE symposium on security and privacy (SP)*, pages 987–1004. IEEE, 2016.

[10] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. Can't touch this: Software-only mitigation against rowhammer attacks targeting kernel memory. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 117–130, Vancouver, BC, 2017. USENIX Association.

[11] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 2204–2206, New York, NY, USA, 2018. ACM.

[12] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[13] Joseph Clements and Yingjie Lao. Hardware trojan attacks on neural networks, 2018.

[14] Ilias Diakonikolas, Gautam Kamath, Daniel M. Kane, Jerry Li, Jacob Steinhardt, and Alistair Stewart. Sever: A robust meta-algorithm for stochastic optimization, 2018.

[15] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Grand pwning unit: accelerating microarchitectural attacks with the gpu. In *Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU*, page 0. IEEE, 2018.

[16] Sanjay Ghemawat. Tcmalloc : Thread-caching malloc, 2018.

[17] Daniel Gruss, Erik Kraft, Trishita Tiwari, Michael Schwarz, Ari Trachtenberg, Jason Hennessey, Alex Ionescu, and Anders Fogh. Page cache attacks. *arXiv preprint arXiv:1901.01161*, 2019.

[18] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 245–261. IEEE, 2018.

[19] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 300–321. Springer, 2016.

---

[10] http://hpcc.umd.edu

[20] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[25] Barry L Kalman and Stan C Kwasny. Why tanh: Choosing a sigmoidal function. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, volume 4, pages 578–581. IEEE, 1992.

[26] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 361–372. IEEE Press, 2014.

[27] Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. Zebram: Comprehensive and compatible software protection against rowhammer attacks. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 697–710, Carlsbad, CA, 2018. USENIX Association.

[28] Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7), 2010.

[29] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[33] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20, 2015.

[34] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 8. ACM, 2017.

[35] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[36] Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. Nethammer: Inducing rowhammer faults through network requests. *arXiv preprint arXiv:1805.04956*, 2018.

[37] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *Research in Attacks, Intrusions, and Defenses (RAID)*, pages 273–294, 2018.

[38] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 131–138. IEEE Press, 2017.

[39] Jemalloc manual. Jemalloc: general purpose memory allocation functions. http://jemalloc.net/jemalloc.3.html, 2019.

[40] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles A Sutton, J Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 8:1–9, 2008.

[41] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*, pages 722–729. IEEE, 2008.

[42] Minghai Qin, Chao Sun, and Dejan Vucinic. Robustness of neural networks against storage media errors, 2017.

[43] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[44] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1–18, Austin, TX, 2016. USENIX Association.

[45] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. Ares: A framework for quantifying the resilience of deep neural networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

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

[48] Mark Seaborn and Thomas Dullien. Exploiting the dram rowhammer bug to gain kernel privileges.

[49] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6106–6116. Curran Associates, Inc., 2018.

[50] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015.

[51] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield. Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4241–4247. IEEE, 2017.

[52] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[53] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

[54] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems*, pages 3517–3529, 2017.

[55] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning FAIL? generalized transferability for evasion and poisoning attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1299–1316, Baltimore, MD, 2018. USENIX Association.

[56] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[57] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the perils of security-oblivious energy management. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1057–1074, Vancouver, BC, 2017. USENIX Association.

[58] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating software mitigations against rowhammer: a surgical precision hammer. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 47–66. Springer, 2018.

[59] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating software mitigations against rowhammer: a surgical precision hammer. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 47–66. Springer, 2018.

[60] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer attacks over the network and defenses. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 213–226, Boston, MA, 2018. USENIX Association.

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

[62] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1675–1689. ACM, 2016.

[63] Bolun Wang, Yuanshun Yao, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. With great training comes great vulnerability: Practical attacks against transfer learning. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1281–1297, Baltimore, MD, 2018. USENIX Association.

[64] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7675–7684. Curran Associates, Inc., 2018.

[65] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[66] Jidong Xiao, Zhang Xu, Hai Huang, and Haining Wang. Security implications of memory deduplication in a virtualized environment. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12. IEEE, 2013.

[67] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 19–35, Austin, TX, 2016. USENIX Association.

[68] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[69] Yan Zhou, Murat Kantarcioglu, and Bowei Xi. Breaking transferability of adversarial samples with randomness, 2018.

# Appendix

## A  Network Architectures

We use 19 DNN models in our experiments: six architecture and their variants. Table 7 describes two architectures and their six variations for MNIST. For CIFAR10, we employ the base architecture from [55] that has four convolutional layers and a fully-connected layer, and we make three variations of it. CIFAR10-AlexNet[11] and CIFAR10-VGG16[12] are from the community. For ImageNet, we use the DNN architectures available from the Internet[13]. In Sec 6.2, we employ two networks (8-bit quantized[14] and binarized versions of MNIST-L5) from the community[15] with adjustments.

## B  The Vulnerability Using Different Criterion

We examine the vulnerable parameter ratio (vulnerability) using the different RAD criterion with 15 DNN models. Our results are in Figure 12. Each figure describe the vulnerable parameter ratio on a specific RAD criterion; for instance, in MNIST-L5, the model has 40% of vulnerable parameters that cause [RAD > 0.5], which estimates the upper bound of the blind attacker. In MNIST, CIFAR10, and two ImageNet models, the vulnerability decreases as the attacker aims to inflict the severe damage; however, in ImageNet, ResNet50, DenseNet161, and InceptionV3 have almost the same vulnerability ($\sim$50%) with the high criterion [RAD > 0.8].

## C  Hyper-parameters for Training

In our experiments, we use these hyper-parameters:

- **MNISTs.** For MNIST models, we use: SGD, 40 epochs, 0.01 learning rate (lr), 64 batch, 0.1 momentum, and adjust learning rate by 0.1, in every 10 epochs.

- **CIFAR10s.** For Base models we use: SGD, 50 epochs, 0.02 lr, 32 batch, 0.1 momentum, and adjust lr by 0.5, in every 10 epochs. For AlexNet, we use: 300 epochs,

---

[11] https://github.com/bearpaw/pytorch-classification/blob/master/models/cifar/alexnet.py
[12] https://github.com/kuangliu/pytorch-cifar/blob/master/models/vgg.py
[13] https://github.com/pytorch/vision/tree/master/torchvision/models
[14] https://github.com/eladhoffer/quantized.pytorch
[15] https://github.com/jiecaoyu/XNOR-Net-PyTorch

Table 7: **8 Network Architectures for MNIST.** We take the two baselines (Base and LeNet5) and make four and two variants from them, respectively. Note that we highlight the variations from the baselines in red color.

| Base | | Base (Wide) | | Base (Dropout) | | Base (PReLU) | |
|---|---|---|---|---|---|---|---|
| **Layer Type** | **Layer Size** | **Layer Type** | **Layer Size** | **Layer Type** | **Layer Size** | **Layer Type** | **Layer Size** |
| Conv (R) | 5x5x10 (2) | Conv (R) | 5x5x20 (2) | Conv (R) | 5x5x10 (2) | Conv (P) | 5x5x10 (2) |
| Conv (R) | 5x5x20 (2) | Conv (R) | 5x5x40 (2) | Conv (-) | 5x5x20 (2) | Conv (P) | 5x5x20 (2) |
| - | - | - | - | Dropout (R) | 0.5 | - | - |
| FC (R) | 50 | FC (R) | 100 | FC (R) | 50 | FC (P) | 50 |
| - | - | - | - | Dropout (R) | 0.5 | - | - |
| FC (S) | 10 | FC (S) | 10 | FC (S) | 10 | FC (S) | 10 |

| Base (D-BNorm) | | LeNet5 [33] | | LeNet5 (Dropout) | | LeNet5 (D-BNorm) | |
|---|---|---|---|---|---|---|---|
| **Layer Type** | **Layer Size** | **Layer Type** | **Layer Size** | **Layer Type** | **Layer Size** | **Layer Type** | **Layer Size** |
| Conv (-) | 5x5x10 (2) | Conv (R) | 5x5x6 (2) | Conv (R) | 5x5x6 (2) | Conv (-) | 5x5x6 (2) |
| BatchNorm (R) | 10 | - | - | - | - | BatchNorm (R) | 6 |
| - | - | MaxPool (-) | 2x2 | MaxPool (-) | 2x2 | MaxPool (-) | 2x2 |
| Conv (-) | 5x5x20 (2) | Conv (R) | 5x5x16 (2) | Conv (R) | 5x5x16 (2) | Conv (-) | 5x5x16 (2) |
| BatchNorm (R) | 20 | - | - | - | - | BatchNorm (R) | 16 |
| - | - | MaxPool (-) | 2x2 | MaxPool (-) | 2x2 | MaxPool (-) | 2x2 |
| - | - | Conv (R) | 5x5x120 (2) | Conv (R) | 5x5x120 (2) | Conv (R) | 5x5x120 (2) |
| - | - | - | - | - | - | BatchNorm (R) | 120 |
| Dropout (R) | 0.5 | - | - | Dropout (R) | 0.5 | Dropout (R) | 0.5 |
| - | - | MaxPool (-) | 2x2 | MaxPool (-) | 2x2 | MaxPool (-) | 2x2 |
| - | - | Conv (R) | 5x5x120 (1) | Conv (R) | 5x5x120 (1) | Conv (R) | 5x5x120 (1) |
| FC (R) | 50 | FC (R) | 84 | FC (R) | 84 | FC (R) | 84 |
| Dropout (R) | 0.5 | - | - | Dropout (R) | 0.5 | Dropout (R) | 0.5 |
| FC (S) | 10 | FC (S) | 10 | FC (S) | 10 | FC (S) | 10 |
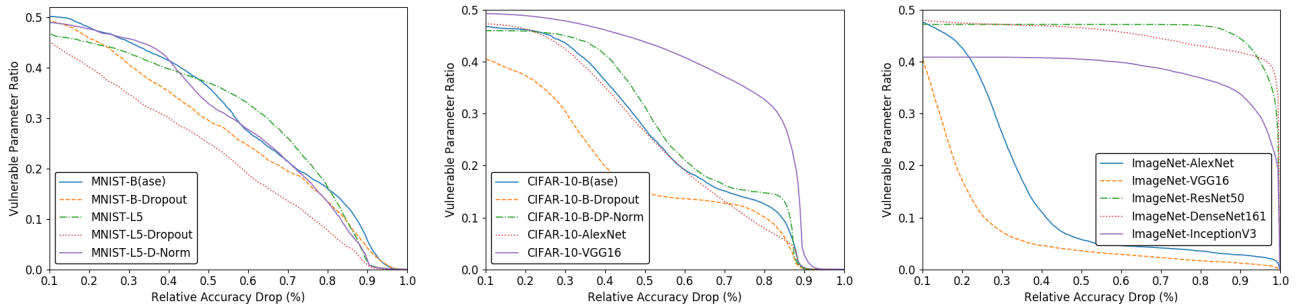


Figure 12: **The vulnerability of 15 DNN models using different criterions.** We plot the vulnerable parameter ratio based on the different RADs that an attacker aims; 5 from MNIST (left), 5 from CIFAR10 (middle), and 5 from ImageNets (right).

0.01 lr, 64 batch, 0.1 momentum, and adjust lr by 0.95, in every 10 epochs. For VGG16, we use: 300 epochs, 0.01 lr, 128 batch, 0.1 momentum, and adjust lr by 0.15, in every 100 epochs.

- **GTSRB.** We fine-tune VGG16 pre-trained on ImageNet, using: SGD, 40 epochs, 0.01 lr, 32 batch, 0.1 momentum, and adjust lr by 0.1 and 0.05, in 15 and 25 epochs. We freeze the parameters of the first 10 layers.

- **Flower102.** We fine-tune ResNet50 pre-trained on ImageNet, using: SGD, 40 epochs, 0.01 lr, 50 batch, 0.1 momentum, and adjust lr by 0.1, in 15 and 25 epochs. We freeze the parameters of the first 10 layers.