Resilient Neural Networks at the Edge: Uncovering and Mitigating Bit-Flip Vulnerabilities in Full-Precision and Quantized DNNs

by

Mahmoud Abumandour

B.Sc., Mansoura University, 2022

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

Declaration of Committee

Name:	**Mahmoud Abumandour**
Degree:	**Master of Science**
Thesis title:	**Resilient Neural Networks at the Edge: Uncovering and Mitigating Bit-Flip Vulnerabilities in Full-Precision and Quantized DNNs**
Committee:	**Chair:** Tianzheng Wang Assistant Professor, Computing Science
	Alaa R. Alameldeen Supervisor Associate Professor, Computing Science
	Arrvindh Shriraman Committee Member Associate Professor, Computing Science
	Zhenman Fang Examiner Assistant Professor, Engineering Science

Abstract

Deep Neural Networks (DNNs) are vulnerable to attacks that reduce accuracy and impact critical applications that rely on their performance. Bit-flip attacks (BFA) enable an attacker to identify a small number of bits that, when flipped, could severely degrade DNN model accuracy. This thesis studies the impact of random and targeted BFA on DNN model accuracy for edge devices. We propose a simple software mechanism to limit an attack's impact on full-precision DNN models. We uncover a vulnerability in quantized DNN models where a few critical model parameters use a higher-precision representation. To our knowledge, we demonstrate the first semi-black box BFA that degraded the accuracy of a quantized DenseNet121 model from 85% to the level of a random guesser (10%) by only flipping 14 out of more than 58 million bits. To address this vulnerability, we propose a software redundancy mechanism that can effectively defend against random and targeted bit-flip attacks with an average of 0.3% performance overhead and 8.2% storage overhead in the worst case.

Keywords: Machine Learning Security; Bit-Flip Attacks; Deep Learning; Quantization; Hardware Security; Rowhammer

Acknowledgements

I want to extend my deepest gratitude to my advisor, Professor Alaa Alameldeen, whose guidance and unwavering support have been instrumental throughout this journey. Your insights and encouragement have profoundly shaped my research and academic growth.

I am also thankful to my Memory Architecture Research Group lab mates at SFU, with whom I have had many thought-provoking and insightful conversations that have impacted this work in many ways.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep neural networks (DNNs) are the foundation of a multitude of artificial intelligence applications such as image classification [13], speech recognition [52], autonomous driving [79], image captioning [3, 44], and medical diagnosis [14]. DNNs are deployed on many systems of different capabilities, from small, limited consumer devices to large, high-performance computer clusters. It is becoming more popular to run DNNs on consumer devices to provide real-time intelligent insights to the user. As state-of-the-art DNNs are computationally intensive, deploying them efficiently on small edge devices requires optimization and down-scaling. Model quantization is a widely employed technique as it substantially reduces energy consumption and memory footprint while preserving model accuracy [31, 28, 27]. The increasing dependence on edge devices powered by DNNs [66, 51, 33] underscores the importance of protecting the models and their internal data integrity. This is especially important, considering that sensitive user data is often involved. Therefore, ensuring the reliability and confidentiality of DNN models and their internal data becomes crucial for ensuring the secure functionality of these devices in everyday use.

Edge devices operate in diverse environments without integrity or safety guarantees for stored data. Hence, such devices are vulnerable to hardware fault injection attacks like Rowhammer [35]. Such mechanisms have been successfully used to launch attacks against both full-precision and quantized DNNs, which can compromise the model by modifying a few parameters in memory [84, 62, 48].

Hong et al. [23] show that for full-precision 32-bit floating point (FP32) models, the most considerable accuracy degradation occurs when flipping bits that increase the magnitude of parameters to extreme values (i.e., the most significant bits in the exponent field) which results in exploded, wrong model outputs. Conversely, quantized DNNs are recognized for resilience against random bit-flip attacks (BFA) as parameters are represented using small-bit widths [62]. This limits the range of possible parameter values, reducing the severity of changing model parameters via bit flips.

This thesis presents two key findings. Firstly, we show that full-precision models can be made more resilient against bit-flip attacks using a simple range-checking mechanism that ensures model parameters are within an acceptable range of values. Secondly, we show that quantized models have nontrivial amounts of data susceptible to random BFA, which can be exploited to launch an attack with a high success probability while having limited knowledge about the system. In quantized models, the majority of parameters are stored in a low-width representation, e.g., 8-bit integer values. However, some critical model parameters, such as bias vectors and batch normalization parameters, are quantized to a higher-precision integer representation, typically 32-bit integers [38, 65, 31, 39]. This approach allows the model to maintain high accuracy while still reducing its storage requirements since the high-precision parameters constitute only a tiny portion of the overall model size.

We present a BFA that targets those highly vulnerable regions and show that it's highly effective in degrading DNN models for seven different DNN architectures that perform different tasks. For example, we show that an attacker can successfully degrade DenseNet121[26] trained on CIFAR10 from 85% accuracy to 10%, turning it into a random guesser by flipping 10 bits on average. We also showcase the impact of our proposed attack against the GPT2 Large Language Model [61]. We show that the model outputs meaningless text in the presence of a small number of bit-flips.

Many recent hardware and software mechanisms have been proposed to defend against Rowhammer attacks, such as [17, 57, 43, 80, 67]. However, such mechanisms require nontrivial hardware changes and incur nontrivial performance overheads for low Rowhammer thresholds. Therefore, these hardware proposals are more suitable for larger systems than for resource-constrained edge devices, which are the focus of our proposal.

To defend quantized DNN models on edge devices against BFA, we propose and evaluate a software mechanism that uses *triple modular redundancy* (TMR) to protect and rectify sensitive model data. We show we can mitigate the attack with low performance and storage overheads. Our experimental results demonstrate that the average performance overhead for our software mechanism is 0.3% across many models. We also show that the storage overhead is 3.7% of model size on average (8.2% in the worst case).

In this thesis, we make the following contributions:

- We propose and evaluate a simple software range-checking mechanism to mitigate BFA against FP32 models.
- We uncover a vulnerability that stems from modern DNN quantization methods, allowing attackers to identify vulnerable regions in a model without time-consuming testing of model parameters.
- To our knowledge, we propose the first semi-black box BFA against quantized DNNs, successfully degrading various models to random guesser-level by flipping a small number of bits in high-precision tensors. The attack requires limited knowledge about the model and the underlying system.

- We propose a software protection mechanism against this attack and show that it can effectively defend quantized DNNs while incurring negligible performance and storage overheads.

Chapter 2

Background

This chapter provides an overview of existing BFA vectors. We then explain how DNNs are quantized and organized in memory. Next, we overview Rowhammer, a powerful hardware fault-injection technique. We then review how Rowhammer attacks are mounted. Finally, we discuss methods for hardening and safeguarding data in memory systems based on redundancy and other hardware-based Rowhammer mitigation mechanisms.

2.1 Bit-Flip Attacks

BFA against DNNs can be categorized into targeted and untargeted attacks. Targeted attacks aim to stealthily deceive the DNN into mispredicting specific inputs while maintaining the prediction accuracy for other input samples, allowing the attacker to control the misprediction behaviour of the model in a concealed manner. For instance, T-BFA [63] is shown to divert a ResNet-18 model trained on the ImageNet dataset to mispredict *all* the testing images from one source class to a target class selected by the adversary without significantly affecting the overall model accuracy by flipping 27 bits. In contrast, the objective of untargeted BFA is to degrade the overall inference accuracy of the model by flipping a small number of bits in the model parameters.

Full-precision model parameters are highly vulnerable to simple untargeted BFA. Stored in the IEEE-754 single floating point representation, perturbations in the most significant exponent bits can change a parameter to an extreme value, drastically changing the model's outputs. Prior work [23] shows that, for various models, flipping the most significant exponent bit in any of 40% of the parameters can significantly degrade model accuracy. For quantized models, random bit-flips do not substantially change parameter values and have an insignificant effect on model accuracy. Hence, the conventional wisdom is that BFA is not as easily executable on quantized models as it is for FP32 models. To launch a successful BFA on a quantized model, the attacker needs perfect white-box access to the model and its parameters to be able to search for specific bit chains that incur the most damage on the model.

The iterative approach followed by most state-of-the-art attacks involves searching for the bit locations in the parameter matrix that would incur the maximum increase in the calculated loss using the testing set x:

$$\Theta^{i+1} = \underset{\hat{\theta} \in \{\hat{\Theta}^i\}}{\operatorname{argmax}} \left[\mathcal{L}(f(x;\hat{\theta})) - \mathcal{L}(f(x;\Theta^i))\right] \tag{2.1}$$

In Equation 2.1, $\mathcal{L}$ is the loss function, Θ^i is the parameter matrix after iteratively flipping i bits, and $\{\hat{\Theta}^i\}$ is the set of possible parameter matrices after flipping a single bit starting with Θ^i. Using a sophisticated bit-search algorithm necessitates that the attacker has full white-box access to the model (including all the parameter values and a batch of the training or the testing sets) and the underlying system to run the search offline before attempting to induce a bit-flip which is not always possible. The majority of prior works manage to degrade a quantized DNN to a random guesser using only a tiny number of bit-flips using progressive vulnerable bit-identifying algorithms while assuming that the attacker has open access to model architecture, values of all the parameters, and a batch of training or testing data [62, 84].

2.2 DNN Model Quantization

State-of-the-art DNN models require substantial storage and computation, making them unsuitable for small, computationally limited devices. One widely used technique for deploying large models on small devices is quantizing model parameters into a lower bit-width representation.

Integer-only Quantization

Model parameters are commonly quantized to integer-only values as integer arithmetic is far less energy-consuming than operations on floating-point parameters.

To convert a real-valued parameter $r \in \mathbb{R}$ to a quantized representation q, an affine quantization scheme is commonly used as it allows integer-only arithmetic without having to de-quantize the parameters to their original representation. One popular affine quantization scheme, described in [31], is as follows.

$$q = Q(r) = \text{round}(\frac{r}{S}) + Z \tag{2.2}$$

where Q is the function that maps real values to quantized integer values; S is the scaling factor, a real-valued number responsible for defining the distance between adjacent quantization levels; and Z is the integer zero-point. The choice of $Z = 0$ is known as *symmetric quantization*, which allows the mapping of $r = 0$ to $q = 0$ without a quantization error. Symmetric quantization is widely used as zero-padding operations are commonly used

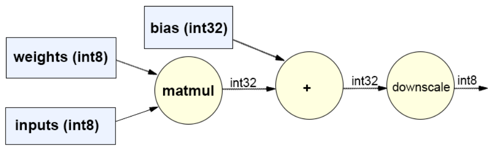


Figure 2.1: The structure of a fully connected layer. Operations are done completely in the integer domain. Down-scaling from 32-bit to 8-bit integers is carried out after the fully connected operation before applying the activation function.

in DNNs. S and Z are called the quantization parameters and are maintained separately for each tensor.

In symmetric quantization, S is calculated as follows:

$$S = \frac{\beta - \alpha}{2^b - 1} \tag{2.3}$$

where b is the bit width of quantized values and α and β are the FP32 range parameters, beyond which values are clipped to q_{min} and q_{max}.

Static vs. Dynamic Quantization

Values for α and β can be determined either *statically* before inference or *dynamically* for every input sample. Since weight matrices typically do not change during inference, range detection and quantization are applied before running inference. Jacob et al. [31] uses $\alpha = w_{min}$ and $\beta = w_{max}$. Activations differ for each input sample, so they are often quantized dynamically.

Dynamic quantization requires computing statistics on activations during inference to decide on an appropriate range, which means that it can be more accurate at the cost of computation overhead.

Static quantization computes range parameters before inference, meaning a single range will be used for all inputs. This results in lower accuracy for out-of-distribution input samples but does not impose any computational overhead. One approach to finding range parameters during static quantization is to run inference for various inputs and averaging the seen ranges [31]. Choukroun et al. [9] propose choosing the range parameter that minimizes the mean square error between the quantized and unquantized values.

Quantizing Standard Operations in DNNs

A common operation during DNN inference is multiplying the matrices of weights and activations and then adding the result to the bias term (also known as the fully connected layer). Weights and activations are commonly quantized to 8-bit integer representation.

Hence, intermediate results of element-wise multiplications must be accumulated in 32-bit integer accumulators. For this reason, biases are either quantized to 32-bit integers or not quantized at all in all integer quantization schemes we have surveyed [38, 65, 31, 39]. For example, in PyTorch 2.3 [58], biases are quantized to 32-bit integers in both static and dynamic post-training quantization (PTQ) and not quantized at all in quantization-aware training (QAT). Furthermore, biases only make up a tiny portion of the overall model size while having a high impact on model accuracy, so maintaining a high precision for biases is vital to preserving model accuracy after quantization. Other important parameters, such as batch normalization, are kept in a high-precision format. Figure 2.1 shows the structure of a quantized fully-connected layer.

Another standard operation used in modern DNNs is Batch Normalization [30]. It helps stabilize and accelerate training by normalizing each layer's inputs to have zero mean and unit variance. This reduces the dependency on initial weight values and allows for higher learning rates, accelerating training without the risk of divergence. Quantizing batch normalization parameters to a low-width representation can be detrimental to learning. The parameters, including mean, variance, and learnable scale and shift, require high precision and an extensive parameter range to reflect the data distribution and ensure network stability. High precision is essential because the computation of these parameters involves reciprocation, square root, and sum of product operations, which are sensitive to numerical inaccuracies. Quantization to lower precision can introduce significant errors, leading to improper normalization, training instability, slower convergence, and reduced model performance. Typically, Batch Normalization parameters are either quantized to a higher-precision integer representation [31] or left in their original floating point representation [86]

2.3 DRAM Organization and Operation

The predominant technology in memory systems is *Dynamic Random Access Memory* (DRAM). It's organized in a hierarchy of channels, dual in-line memory modules (DIMMs), ranks, banks, and cells. Cells are the indivisible storage unit in DRAM, containing a capacitor to store the charge denoting a single bit and a transistor to control access to the cell value. Cells are organized in columns and rows, which make up *banks*. Banks are grouped to form *ranks* (typically, each side of the DIMM module is a rank). Each DIMM is inserted into a channel, connecting the memory module to the CPU and its memory controller. DRAM is usually accessed on the granularity of a *row*, which is 8KiB, by *copying* it into a per-bank *row buffer*. After bringing the correct row into the row buffer, the CPU can request data at the granularity of 64-bit words served from the row buffer so long as it has not been replaced by another row.

Memory controllers use an addressing function to map each physical address to a specific location in memory (channel, DIMM, rank, bank, row, and column). Those functions are undocumented for various CPU manufacturers but can be reverse-engineered using physical hardware or software methods [59, 68]. A standard mapping function is choosing bits in the physical address and XORing them together to minimize bank conflicts. Seaborn [68] reports that on his setup with 8192 MB physical memory over two DIMMs, the mapping is shown in Table 2.1. Each DIMM has two ranks, eight banks in each rank, 32768 (2^{15}) rows per bank, and each row is 8 Kilobytes.

Table 2.1: Address mapping function reported in [68]

Bit index	**Usage**
0-5	The six least-significant bits in the byte index into the row. Used as the cache block byte index (for a 64-bit cache block)
6	Channel selector
7-13	High order bits of the byte index into the row
14-16	XOR'd with the three least significant bits of the row index to select the bank
17	Rank selector
18-32	Row index in the bank
33+	May be used if the physical memory does not start from address 0

2.4 Memory Fault Injection using Rowhammer

DRAM is vulnerable to Rowhammer [35], a memory fault injection mechanism caused by DRAM disturbance errors. This vulnerability arises from repeatedly accessing a DRAM row, intensifying the electromagnetic coupling between DRAM cells and accelerating charge leakage from adjacent rows. The minimum number of required DRAM row activations to induce a bit-flip is called the *Rowhammer Threshold (T_{RH})*. Due to DRAM scaling and continuous shrinkage of DRAM cells, this threshold has been decreasing, making DRAM chips more susceptible to errors. Rowhammer was demonstrated as a security threat against systems of different varieties, including embedded systems [15], mobile systems [74] and servers [46]. High Bandwidth Memory (HBM), due to its dense architecture and close proximity of cells, is also particularly vulnerable to Rowhammer [55], exacerbating the challenge of mitigating such attacks in modern memory technologies. Flipping a single bit in the victim's address space typically takes a long time, and bit-flips are transient (i.e., on disk reload, bit-flips are rectified). Hence, attackers must identify a small subset of highly important bits to target and cause to flip. Attacks that utilize Rowhammer have been used to leak confidential data and escalate privileges [69, 17, 74, 85]. Rowhammer-based methods have been the principal hardware vulnerabilities exploited to launch attacks against systems running DNNs [84, 63].

There are two main variants of Rowhammer: single- and double-sided. Single-sided Rowhammer repeatedly accesses a single aggressor row and expects flips to happen in the two adjacent rows. Double-sided Rowhammer requires achieving a *sandwich memory layout* wherein the attacker hammers the two adjacent rows and expects bit flips in the middle row. Double-sided Rowhammer induces higher pressure on the target row, amplifying charge leakage and inducing more bit flips [4]. However, it requires virtual-to-physical and physical-to-bank mapping knowledge, which may not always be obtainable.

2.5 Mounting Rowhammer

Mounting Rowhammer can be tricky even if the DRAM chip is vulnerable, as it breaks the abstractions provided by the hardware and the operating system. This section demonstrates the three obstacles an attacker faces in mounting a successful Rowhammer attack.

2.5.1 Uncached Memory Access

The attacker must race against the DRAM refresh time to flip successfully before restoring the charge in the target cells. The x86 ISA provides an explicit cache flush instruction `clflush`, which the attacker can repeatedly use to remove the hammered data from the cache. Another approach is to repeatedly access addresses that map to the same cache set, thereby enforcing repeated cache evictions. x86 also provides *non-temporal* access instructions that inherently don't bring the accessed data into the cache.

2.5.2 Virtual-to-Physical Memory Mapping

One of the challenges against flipping bits in the victim's memory is to position memory pages in such a way that allows the hammering of the attacker's data to affect the victim's bits. The first practical version of a Rowhammer attack [69] used the `/proc/PID/pagemap` provided for Linux systems, providing complete information about virtual-to-physical page mapping. Unprivileged access to such an interface has been prohibited since Linux kernel version 4.0 was introduced [70]. Another approach is to use the huge pages feature available to some x86 systems, which gives the user process a large, contiguous physical memory block, providing sufficient mapping information for the attacker to mount the attack.

2.5.3 Memory Layout Massaging

For an attacker to induce bit-flips in the victim's data, the target data must be carefully positioned in a place the attacker can affect by repeatedly accessing adjacent data. The attacker, therefore, needs to influence the system to allocate the target data in such a location.

One approach used by [64] exploits *memory deduplication*[1] to trick the OS into mapping a victim page and an attacker page to the exact physical page containing Rowhammer-vulnerable bits at desired offsets. However, even though it has been demonstrated as a reliable method for memory massaging, it is not an on-by-default feature and is turned off on many systems.

Another approach, *memory waylaying*, is described in [20]. File page data is cached in DRAM in most operating systems. These cached pages are considered free, available memory, as they can be evicted anytime. When one of the page cache pages is evicted, it is brought back into a random page in memory in both Linux and Windows. The attacker can repeatedly influence the system to evict the target page until it is placed on an attacker-chosen page. Waylaying depends on a prefetching side-channel described in [21] to detect when the target page is in a suitable location from the attacker's perspective.

Kwong et al. [42] describe an approach that exploits the *Linux buddy allocator* to allocate a contiguous block of 2 MiB of physical memory. The attacker must know when the victim will allocate the target page (after n pages of allocations). The attacker would free a page with flippable bit locations and then free other $n-1$ pages. Those pages would go back into the pool in a last-in-first-out fashion. After that, the attacker would invoke the victim, which will cause the target page to be placed in the first-freed page (the one with vulnerable locations).

2.6 Defences against Bit Faults

Systems with severe reliability requirements can employ defence mechanisms to safeguard sensitive data against failures. Various software and hardware mechanisms have been proposed to defend against bit errors. Herein, we describe some of these approaches.

2.6.1 Error-Correcting Codes (ECC)

ECC is a technique commonly used to guarantee reliability against errors in memory data symbols. DRAM chips with ECC implement *binary linear block code* schemes to encode data blocks into *codewords*. A codeword consists of r data bits and k parity-checking bits. The number of parity bits k controls the tradeoff between reliability guarantees and storage and performance overheads. For instance, the DDR4 standard [32] specifies the codeword size of 72 bits, with $r = 64$ and $k = 8$. Other systems implement more complex ECC schemes inside the memory controller [29], such as *chipkill* [12]. Chipkill distributes ECC bits across memory chips, enabling regular memory operation even when an entire DRAM chip fails. Although ECC (Error-Correcting Code) provides data integrity guarantees, recent work has

[1]Memory deduplication is a kernel mechanism that merges physical pages with identical content and marks the virtual pages as copy-on-write

shown that systems implementing ECC remain susceptible to the Rowhammer attack [10]. This attack exploits the timing difference between accessing a correct codeword and one with a bit error that is detected and corrected by ECC and then induces more bit flips in the same row. This vulnerability has been demonstrated even in ECC schemes that are capable of correcting multiple bit errors, including advanced techniques like Chipkill. Therefore, while ECC enhances reliability against bit errors, it is insufficient as a standalone defense against Rowhammer attacks.

2.6.2 Rowhammer Defences

Various hardware and software mitigation mechanisms have been proposed to defend against Rowhammer. Typically, a Rowhammer defence involves an access-tracking mechanism and a corrective action that is taken when a potential hammering attempt is detected. In Target Row Refresh (TRR) and similar techniques [17, 57, 43], access tracking is done using counters that track the number of accesses for a potential *aggressor row*, and when the number of accesses exceeds a predefined threshold, adjacent potential *victim rows* are refreshed. Those mechanisms require hardware modifications and may require extra storage impervious to Rowhammer to store the counters [57]. Furthermore, systems with victim-refreshing protection are still susceptible to more sophisticated attacks such as Half-Double [36], which does not target the immediately adjacent rows.

More recent defences, such as AQUA [80] and SRS [67], apply the corrective action on the aggressor row by either swapping it to a random location or quarantining it, breaking its spatial proximity with the victim row. While those aggressor-row-centric mechanisms are resilient against sophisticated hammering patterns, they incur a prohibitive performance overhead for low T_{RH}, projected to decrease aggressively in the upcoming years. Table 2.2 shows the measured threshold for different DRAM generations with labels *old* and *new* referring to the same generation classified by the manufacturing date [34]. While those solutions provide strong guarantees against Rowhammer, the overhead and the required hardware modifications may not be practical for resource-constrained edge devices and are usually considered for large and more capable systems.

2.6.3 Fault Tolerance using Replication

N-modular redundancy (NMR) is a standard protection approach for systems with severe security requirements. NRM entails storing N copies of sensitive data in memory. When data is accessed, the system retrieves and compares the N copies simultaneously. A majority vote is conducted among the N copies to determine the correct value, and the most common value is designated as correct. In the event of error detection (i.e., no consensus in the data copies), the system can correct all the stored copies to the designated correct value. While NMR provides high levels of fault tolerance, it can come at the expense of significant performance and storage overhead, especially when dealing with large amounts of sensitive data. For

Table 2.2: Rowhammer threshold over DRAM generations from 2014 to 2021. [34, 35]

DRAM Generation	Rowhammer Threshold
DDR3 (old)	139K
DDR3 (new)	22.4K
DDR4 (old)	17.5K
DDR4 (new)	10K
LPDDR4 (old)	16.8K
LPDDR4 (new)	4.8K

example, if N = 3, the system must store three copies of the data, which triples the storage requirements. Additionally, retrieving and comparing multiple copies of data can slow down memory access times and affect cache locality. As a result, NMR may not be practical for applications that require fast access to large amounts of data.

Chapter 3

Threat Model

The primary attack vectors we discuss are mounted against DNNs quantized to a low-precision integer representation and stored in DRAM. The attacker is co-located with the victim's device either on the same system or through the network, enabling them to induce bit-flips in the victim's memory [46, 72, 82]. The attacker's goal is to degrade model accuracy as much as possible. The attacker cannot tamper with the training process and aims to degrade model accuracy for legitimate input samples (i.e., for experimentation, we use the testing set of the task dataset). The attacker is limited in the number of bit-flips they can induce, given that a single bit-flip can take a long time to cause, and they are non-persistent across system resets as correct model data will be reloaded from the storage.

We assume that the attacker knows the underlying model architecture. This assumption is reasonable as deploying open-source, pre-trained models is common. Even for proprietary models, prior works have extracted the model architecture without any previous knowledge [25]. We assume that the attacker lacks access to the model parameter values. This is a reasonable assumption because open-source models (whose source code is available) often require fine-tuning or training from scratch, which would alter their parameter values as the model is further optimized for its specific task. Also, it's common to embed proprietary models optimized for specific tasks in edge devices, in which case the attacker cannot access the parameters. Prior works show that it is possible to estimate network parameters using power or timing side-channels accurately. However, Xiang et al. [81] show that smaller models (the primary targets for this work) are more challenging to accurately reverse-engineer due to their simplicity and small impact on the overall system power consumption. Other prior works exploit specific design or hardware choices, which may not apply to the device under attack due to the diversity in edge devices hardware [71, 78, 6]. We also assume they can mount Rowhammer, targeting random bits within the high-precision regions of the victim model. We demonstrate how an attacker can use Rowhammer to launch the attack we propose in Section 6.3.

Chapter 4

Experimental Setup

This section describes our experimental setup to evaluate the DNN model's vulnerability to BFA. We describe our benchmarks, datasets, software setup, and the hardware system configuration we modelled.

4.1 Model Architectures and Datasets

We test DNNs of different complexities and sizes suitable for various tasks commonly done on mobile or small edge devices. We use all four benchmarks from the MLPerf Tiny benchmark suite [5] and experiment with larger models for one of the benchmarks. We also evaluate our attack for quantized models against GPT2 [61], a Large Language Model (LLM), which can be used as a text completion model for mobile devices. The MLPerf Tiny Benchmark suite includes DNNs trained for tasks commonly run on mobile and small embedded devices. The MobileNetV1 architecture [24] is trained using the MSCOCO dataset [45] and is used for the Visual Wake Words benchmark, which targets detecting the presence of a person or more in input images. The Keyword Spotting benchmark trains a Decoder-Side Convolutional Neural Network (DS-CNN) [83] using the Google Speech Commands dataset [76] to recognize short phrases using sound input. The Anomaly Detection benchmark trains a fully connected AutoEncoder on machine noises to detect anomalous machines using the ToyADMOS dataset [37]. Finally, the Image Classification benchmark trains a small Residual Neural Network (ResNet-8) [22] on the CIFAR10 dataset [40], which contains 60,000 32x32x3 RGB images categorized into 10 classes. The testing set for CIFAR10 consists of 10,000 images. Similar to prior works, we sample 1,000 images to speed up exhaustive testing. We also experimented with larger models, namely, DenseNet121 and ResNet-50, a more extensive Residual Neural Network, for the image classification benchmark.

4.2 Software Setup

We use TensorFlow [1] version 2.12 to train FP32 models. We then convert FP32 models into an integer 8-bit representation using TensorFlow Lite, which we use as the main testing

Table 4.1: Emulated system configuration

Component	Parameter	Value
CPU	ISA	RISC-V
	Mode	in-order
	Clock	1 GHz
DRAM	Size	256 MB
	Model	LPDDR3
	Clock	800 MHz
L1 ICache / DCache	Cache Size	32 KB
	Associativity	2
	Latency	1 cycle

platform for our bit-flip analysis of quantized DNN models. To test the efficiency of TMR for sensitive data in quantized models, we modify the TensorFlow Lite Micro [11] framework to include redundancy for sensitive parameters and check their integrity at inference time.

4.3 Hardware Platform

To evaluate our proposed protection mechanism for quantized DNNs, we model a small, resource-constrained system core connected to a low-power DRAM memory chip using the gem5 simulator [7]. Table 4.1 shows key configuration parameters for our modelled system.

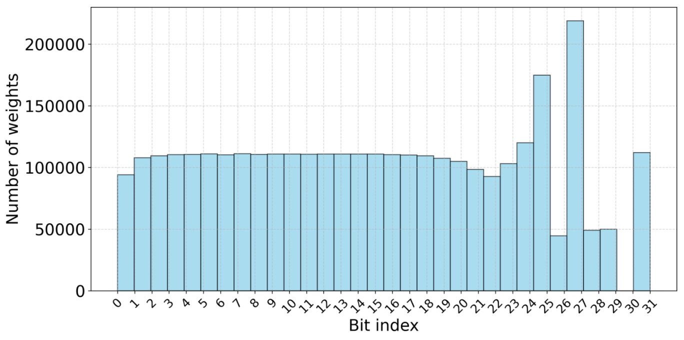
Chapter 5

BFA Against FP32 Models: Analysis and Solution

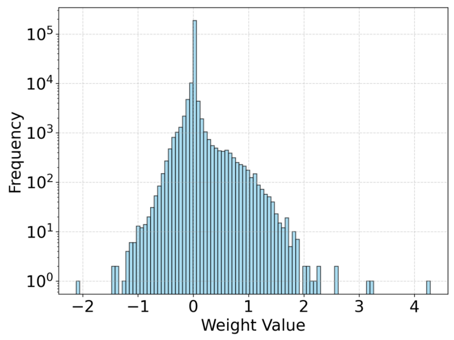
Models stored in the floating point format have been proven to be fragile against BFA due to the vast change a bit-flip could cause if it occurs in the most significant bit locations of the exponent field. Hong et al. [23] show that even a single bit-flip in any of nearly half of the parameters can inflict at least 10% relative degradation in model accuracy without any prior knowledge of the model. We run exhaustive bit-flip testing for six DNNs trained on four tasks and confirm the results presented in [23]. We calculate the percentage of parameters in each model with at least one vulnerable bit. We define a vulnerable bit as a bit that would inflict δ relative accuracy degradation on the model if flipped. For larger models (e.g. DenseNet121 and ResNet-50), testing every bit is too time-consuming as the number of parameters is on the order of millions. To mitigate this, we follow the same approach used in [23] and sample 100,000 parameters at random and limit the tests to the bits in those parameters. This is a sound simplification as even for the models we exhaustively test, the percentage of vulnerable parameters is high and distributed across the model layers.

Most of the risk of BFA against FP32 models comes from flipping the most significant exponent bit of parameters from 0 to 1, resulting in a substantial change in the parameter value, drastically affecting all the calculations that depend on that parameter. Figure 5.1(a) shows the bit distribution for all the parameters of the MobileNetV1 model used for the Visual Wake Words benchmark. In the pristine model, most parameters (99.9941%) have the most significant exponent bit set to 0.

To address the vulnerability in FP32 models, we propose a simple range-checking defence that could be applied to hardware or software. For each model, we store the minimum and maximum parameter values. The parameters are checked against the minimum and maximum parameter values during inference. If a parameter is out of range, the most significant exponent bit is reset to 0. The intuition behind this mechanism is that if a bit-flip attack drastically changes a parameter, this is likely caused by flipping the most significant bits of the exponent. This simple mechanism yields effective results as the range



(a) Bit distribution



(b) Weight Value Distribution

Figure 5.1: (a) shows the bit distribution for the MobileNetV1 FP32 model (trained on the MSCOCO dataset with 85% Top-1 accuracy). A bar at Bit Index = i corresponds to the number of parameters with the ith bit equal to 1. The number of parameters with the most significant exponent bit activated is 13 out of 221,794 (less than 0.0059%). (b) shows the parameter value distribution for the same model.

Table 5.1: Percentage of vulnerable parameters for $\delta = 50\%$ and $\delta = 10\%$.

Model	Accuracy	# Params.	Vuln. params (%)		Vuln. params. in range (%)	
			$\delta = 50\%$	$\delta = 10\%$	$\delta = 50\%$	$\delta = 10\%$
AutoEncoder	81%	100,000	30.61%	64.99%	0.33%	0.59%
DS-CNN	92%	24,908	50.57%	51.95%	2.30%	7.94%
MobileNetV1	85%	100,000	0.24%	9.60%	0.0%	0.44%
ResNet-8	87%	78,666	48.12%	48.74%	0.53%	2.81%
ResNet-50	84%	100,000	13.00%	31.00%	0.11%	0.11%
DenseNet121	88%	100,000	21.09%	32.40%	0.04%	0.09%

of parameters is usually tightly distributed around 0, and flipping the most significant bit in the exponent field will almost certainly take the parameter out of range. Figure 5.1(b) shows the weight distribution of the MobileNetV1 model. We observe that the weight distributions for other models are similar.

Table 5.1 shows the percentage of vulnerable parameters for $\delta = 10\%$ and $\delta = 50\%$ for the model with and without protection. The last column corresponds to the defended model (after range-checking) and shows the number of tested weights that bypass the defence (i.e., the bit-flip does not take the weight out of range **and** degrades the model by δ). The results show that using this simple approach can significantly reduce the risk of BFA against FP32 models. For instance, for $\delta = 50\%$, the percentage of vulnerable parameters is reduced from 30.61% to 0.33% for the AutoEncoder DNN (99.2% reduction in vulnerable parameters). In the worst case, the percentage of vulnerable bits for $\delta = 10\%$ is reduced to 7.94% in the case of DS-CNN. We attribute this to the model's small size and simplicity, so it's more sensitive

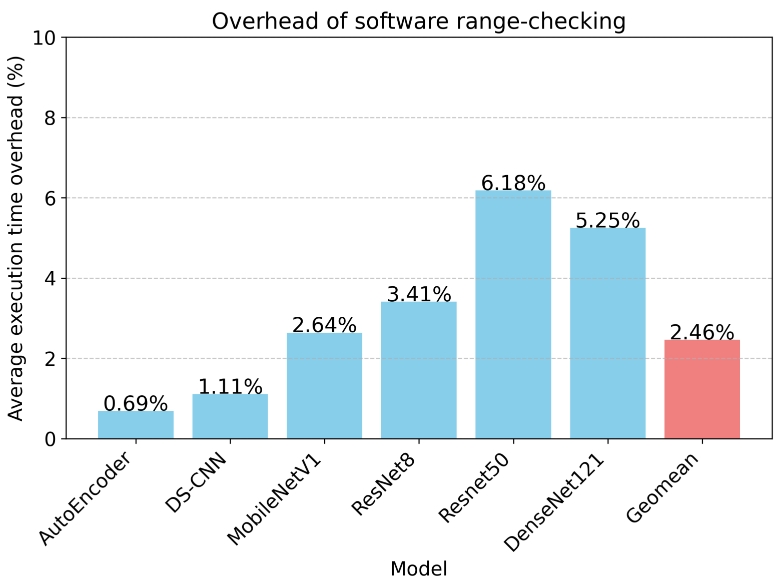


Figure 5.2: Average performance overhead (i.e., additional execution time) of implementing software range-checking.

for bit flips that don't drastically change a parameter. Even in this case, the reduction in the number of vulnerable parameters is 84.72%. Figure 5.2 shows the impact on the performance of using range-checking in software. On average, the incurred overhead in execution time amounts to 2.46%, a tolerable level. However, the overhead grows with the model size.

This range-checking mechanism can be implemented in hardware with modest state overhead and negligible performance overhead for models exhibiting a substantial slowdown since hardware-based range-checking and bit flipping can be much faster than software. The required hardware additions include adding range registers, dedicated comparators to compare model parameters against the range, and simple bit-flipping logic for the most significant bit. We did not explore this hardware mechanism in detail since our software range-checking mechanism is sufficient for resource-constrained edge devices as they primarily run small models. However, a custom hardware solution could be explored in future work.

Per-tensor range checking. Although our findings indicate that this simple software range-checking mechanism adequately safeguards the most vulnerable parameters, it might not be sufficient for the most susceptible models, such as DS-CNN. We explored an alternative approach involving storing per-tensor ranges instead of a global model range. We observe improved outcomes by comparing parameters to their corresponding tensor range, especially for more susceptible models. This approach decreased the percentage of vulnerable parameters for DS-CNN to 1.5% and 6.4% for $\delta = 50\%$ and $\delta = 10\%$, respectively. Other less-vulnerable models were not significantly impacted. However, this strategy introduces a tradeoff as it expands the attack surface, necessitating additional measures to protect many range parameters against tampering. These ranges could be securely stored in on-chip secure memory if available.

Chapter 6

Analysis of BFA Against Quantized DNN Models

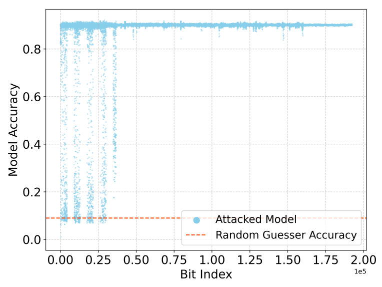
This section analyzes the adverse effect bit-flip attacks can have on quantized models. We then describe a practical Rowhammer attack that a co-located malicious process can mount and compromise the model.

6.1 BFA Against Quantized Models

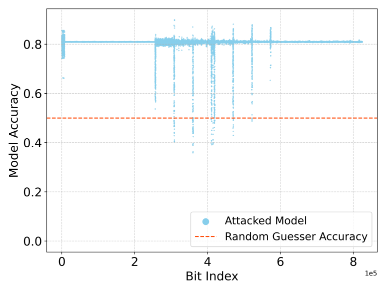
Integer quantization is one of the most common quantization schemes used for DNNs, representing parameters in low-bit-width integers, typically 8-bit integers. The low-precision integer representation necessitates that most of the 2^8 possible values will be used to achieve the best accuracy possible. This means that a bit-flip is unlikely to substantially change the value of a parameter and take it out of the parameter range. Hence, the effect of random bit-flips on quantized DNNs is not significant. To evaluate the resilience of quantized DNNs, we treat the whole model data as a contiguous array of bits in memory, and we exhaustively flip every bit to evaluate the accuracy of the attacked model.

Figure 6.1 shows a complete bit-flip profile of three quantized models (DS-CNN, AutoEncoder, and MobileNetV1). The two main observations from the figure are: 1) a random bit-flip is highly unlikely to affect the model accuracy of a quantized model significantly, and 2) some tiny, contiguous regions are more vulnerable against bit-flips and can degrade the model to worse than a random guesser using a single bit-flip.

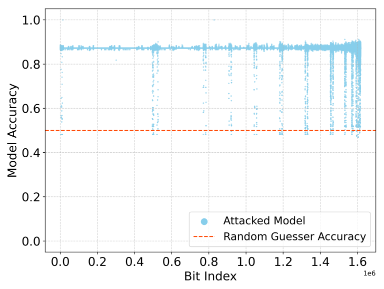
Upon investigating the vulnerable regions, we identified them to be high-precision parameter vectors. In practice, most frameworks either quantize bias tensors and other important low-volume tensors into high-precision integer representation [1, 65, 38, 31, 58] or leave them as FP32 [53]. For instance, high-precision quantization of bias vectors meets a practical need as bias values are added to many layer outputs, making them more sensitive to quantization errors. Furthermore, they constitute a small portion of the model size. Table 7.1 shows the size of high-precision parameter tensors in the six DNNs. Since two



(a) DS-CNN



(b) AutoEncoder



(c) MobileNetV1

Figure 6.1: The accuracy of quantized AutoEncoder (a), DS-CSS (b), and MobileNetV1 (c) against a single bit-flip attack. Each point at Bit Index = i represents the model's accuracy after flipping the ith bit in the parameter matrix viewed as a contiguous array of bits.

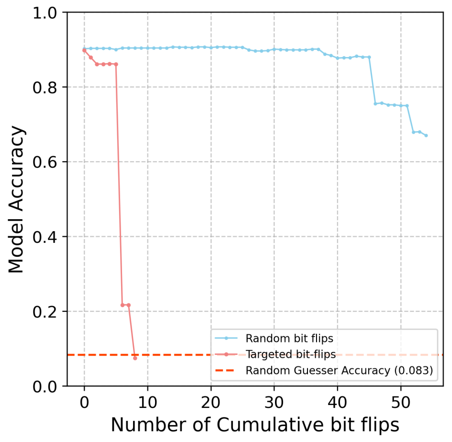
extra copies of those regions are to be stored, the worst-case storage overhead is 8.2% for DS-CNN. We demonstrate a simple yet highly effective attack vector against quantized models, which exploits this characteristic. In the next section, we propose a defence strategy using redundancy in software to protect those sensitive regions.

In scenarios where attackers lack access to exact model parameters, traditional progressive bit-search algorithms become impractical due to their dependence on complete white-box access to the model and its parameter values. Instead, adversaries may resort to flipping as many random bits as possible in model data, which, as we show next, is ineffective in substantially degrading DNNs' accuracy.

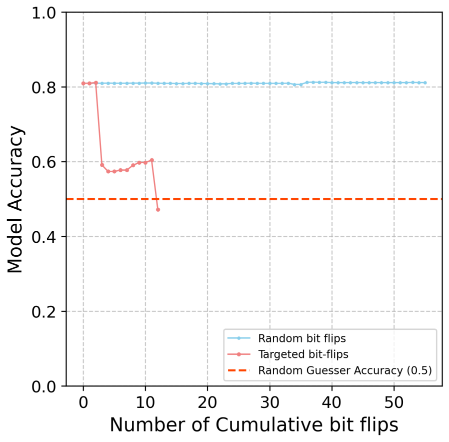
We propose a more effective attack leveraging the limited information available to the adversary. Attackers can significantly impact model performance by identifying and targeting the highly vulnerable, high-precision regions shown in Figure 6.1. Our method involves randomly flipping bits within these regions, exploiting the sensitivity of these parameters to maximize degradation with a high probability. Figure 6.2 shows the results for a single run of a completely random attack versus randomly selecting bits only from high-precision vectors for three models.

In the first strategy, we flip random bits from the whole model. As expected, randomly flipping bits from the entire model is inadequate even after 50 cumulative bit-flips (up to 100 for ResNet-50 and MobileNetV1). At the low end, DS-CNN starts to break at around 40 bits across several runs. Similar to the FP32 version of the model, the small size of the model and its simplicity lead to its breakage relatively quickly after flipping a small number of completely random bits.

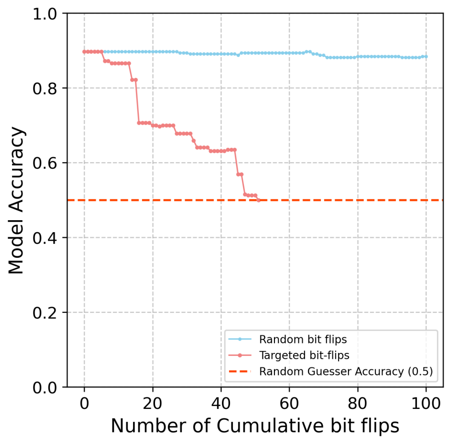
For our proposed attack, we confine the attack's bit flips to the high-precision vectors of the model. This strategy quickly degrades the model to a random guesser in most cases. We have run each attack five hundred times and show the median number of bit-flips within high-precision vectors to turn the model into a random guesser in Table 6.1. This simple attack strategy can inflict significant accuracy degradation on most models using a small



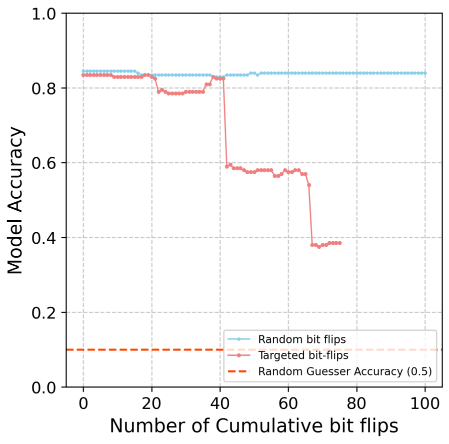
(a) DS-CNN



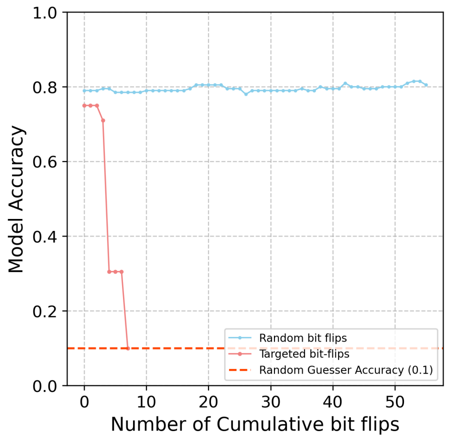
(b) AutoEncoder



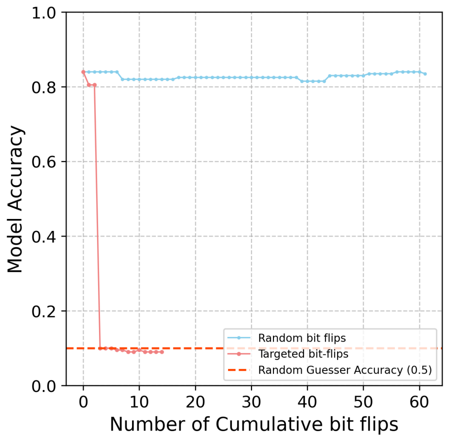
(c) MobileNetV1



(d) ResNet-50



(e) ResNet-8



(f) DenseNet121

Figure 6.2: Testing fully random BFA and random BFA within high-precision vectors.

Table 6.1: Average number of random bit-flips within high-precision vectors to degrade models to a random guesser.

Model	**Pristine Accuracy**	**Median Bit Flips to Random Guesser**
AutoEncoder	81.0%	16
DS-CNN	90.2%	10
MobileNetV1	84.2%	32
ResNet-8	80.0%	7
ResNet-50	84.5%	-
DenseNet121	84.0%	14

number of bit-flips. Even though the attack cannot degrade ResNet-50 to a random guesser using a practical number of bit-flips, we can still inflict significant accuracy degradation on it. For example, the median number of bit-flips required to reduce the accuracy by 10% (i.e., for $\delta = 10\%$) across five hundred runs is 21.

6.2 BFA against quantized LLMs

Table 6.2: GPT2 text completion output for a pristine model versus attacked models with 20 bit-flips in high-precision parameters.

Input Prompt	**Pristine Model Output**	**Attacked Model Output**
"My Trip to Yosemite was"	**My Trip to Yosemite was** a great experience, and I can't wait to go back again to the beautiful park next spring	**My Trip to Yosemite** was a newcom!!!!! (C! (New!! (C!! (New!! (C! (! (! (NewC! (C! (NewC!
"The economy is "	**The economy is** recovering and people are looking for jobs.	**The economy is** this week. The U.S. Treasury issued the first-ever note
"What is the meaning of life?"	**What is the meaning of life?** – "life is the pursuit of the highest ideals,"	**What is the meaning of life?** !" – "! –!!!" – "! –! –!" –!"

To demonstrate the risk of our proposed attack against LLMs, we experiment with GPT2, an LLM commonly used on mobile devices for text completion due to its balanced resource efficiency, making it a suitable choice for deployment despite the constraints of such devices compared to newer, more computationally demanding models. We repeatedly launched 20-bit-flip attacks against it for fifty trials and examined the output manually. We use the same prompts for both the pristine and the attacked models. We observe that

32 trials (out of 50) resulted in the model outputting meaningless text. Table 6.2 shows examples of pristine and attacked model outputs.

6.3 A Practical Rowhammer Attack Against Quantized Models

In Section 6.1, we examined the effect of flipping bits in quantized models' parameters and concluded that high-precision regions are the most vulnerable. Many models we discussed could be degraded to a random guesser using a few bit-flips in those regions. This section describes a practical Rowhammer attack, which a malicious, co-located user can mount and ultimately destroy the model.

The attacker constructs the model in their userspace since they know the model architecture but initialize it with random parameters and run inference once. The attacker monitors memory accesses to know when the vulnerable, high-precision parameters are allocated (after n memory page requests). Next, the attacker needs to allocate n consecutive physical memory pages. Massaging the buddy allocator in Linux has been successfully used to obtain a physically consecutive region of size 2 MiB [42]. The buddy allocator serves blocks of sizes $4096 \cdot 2^n$ that may or may not be contiguous. First, The attacker exhausts the smaller blocks using the `mmap` system call and provides the `MMAP_POPULATE` flag, enforcing the kernel to allocate data immediately instead of the default lazy allocation mechanism. The attack described in [42] uses the `/proc/pagetypeinfo` interface to monitor the available block sizes. However, Linux now requires root privileges to access that interface. Alternatively, the attacker may use the side-channel described in [16] to detect if the allocated chunk is contiguous.

Next, the attacker frees one of the physical pages with vulnerable bit locations. In the attack we describe against quantized models, the attacker does not target specific bits in the high-precision regions but can randomly induce bit flips in them. Hence, any page with flippable locations will suffice. The attacker then frees the remaining $n-1$ pages. Since Linux inserts freed pages into the *page frame cache* in a last-in-first-out fashion, the next nth page to be requested will be allocated in the page with vulnerable locations. The attacker then immediately invokes the victim's process inference. The page with the selected high-precision region will be allocated in the *nth* requested page, which will have vulnerable bit locations.

For a highly effective attack, the attacker may use double-sided hammering to induce the highest number of bit-flips [35]. Double-sided hammering requires the attacker also to know the DRAM address mapping that converts physical addresses to a *(channel, rank, bank, row)* tuple, which can be obtained using the side-channel described in [59].

Chapter 7

Protecting Quantized Models

In Section 6.1, we showed that random BFA against quantized models is ineffective in inducing a significant accuracy degradation. We also uncover a vulnerability in quantized models that exploits the characteristics of how DNNs are quantized in practice. This section proposes using Triple Modular Redundancy (TMR) in software to protect sensitive data. Previously, we have shown that the maximum damage that can be induced by an adversary that cannot run sophisticated bit search algorithms can be achieved by flipping bits in high-precision parameter vectors. Since those vulnerable regions can be determined before deployment and those parameter vectors constitute a small portion of the model, we can afford to use expensive redundancy mechanisms to guarantee their integrity.

One approach is to use DRAM chips that support ECC. ECC memory chips usually apply redundancy to all memory contents and provide simple ECC schemes, such as Single Error Correction, Double Error Detection (SECDED). However, the higher cost of ECC memory is unsuitable for economic edge devices. We have shown that vulnerability is concentrated in small, contiguous regions, so it would be better to provide stronger integrity guarantees for those regions. Furthermore, prior work [10] shows that ECC DRAM chips are not impervious to the Rowhammer attack. Another solution would be to implement secure on-chip vaults to store high-precision parameters. While this would eliminate the risk, the implementation can be complex and inflexible due to the variance of systems running inference. Additionally, storing sensitive parameters in dedicated chips can lead to a single point of failure and would be harder to patch and adapt to newly discovered threats.

We propose using TMR in software to safeguard the high-precision vectors. This entails storing three replicas of the sensitive data. During inference, all three copies are fetched and cross-verified for consistency. Compared to TMR for the whole memory, TMR for the sensitive data only is much more efficient since high-precision vectors constitute tiny portions of the overall model size, as shown in Table 7.1. The worst case for storage overhead is

Table 7.1: The size of high-precision parameter vectors and their portion of total model size

Model	Overall Size	High Precision Parameters Size	%
AutoEncoder	270KB	6.5KB	2.4%
DS-CNN	56KB	2.3KB	4.1%
MobileNetV1	325KB	10.6KB	3.7%
ResNet-8	96KB	1.4KB	1.5%
ResNet-50	26MB	111.3KB	0.4%
DenseNet121	7.3MB	41KB	0.5%
GPT2	124MB	432KB	0.3%

about 8.2% of the model size (for DS-CNN)[1]. Furthermore, as a software solution, it allows full backward compatibility with model files and does not require any modifications to the inference hardware.

To assess the performance implications of our proposed defence, we modified the TensorFlow Lite Micro framework [11] to include extra copies of high-precision parameters. We also modified the inference code to fetch the three copies and conduct a majority vote to determine the correct value. If an inconsistent copy is detected, it is overwritten using the value decided by the majority. We then model a small system in gem5 and run inference to measure the performance overhead. Figure 7.1 shows the performance overhead in normalized execution time for a system running the code with the TMR implementation installed. The geometric mean for performance impact across the six models is 0.3% of the baseline, and in the worst case (DS-CNN), the overhead is 1.36%.

Next, we test the resilience of TMR against our targeted attack. We analyze the worst-case scenario regarding security, wherein the data and the redundant copies are all exposed to the attacker, who can induce bit-flips in any of them. The likelihood of causing bit-flips in at least two of three copies is extremely low, as the Rowhammer-vulnerable bits are rare and scattered throughout the memory. We model this behaviour by flipping random bits uniformly across the three replicas of high-precision vectors, so there is a low probability that at least two bit-flips will land in the same position of the same parameter in the replicas. To stress-test the system, we assume the adversary can launch up to 32 bit-flips. Table 7.2 shows the percentage of runs that induce a 10% relative accuracy drop with and without TMR measured across a thousand runs. TMR consistently provides effective mitigation against the attack across all network architectures. For instance, the attack is successful against DS-CNN and ResNet-8 100.0% of the runs without TMR, and it goes down to 0.4% in both cases when we use TMR at inference time. We observe that there is a slight probability that

[1]Since DS-CNN's sensitive parameters represent 4.1% of the whole model data, creating two additional replicas requires an overhead of 8.2%.

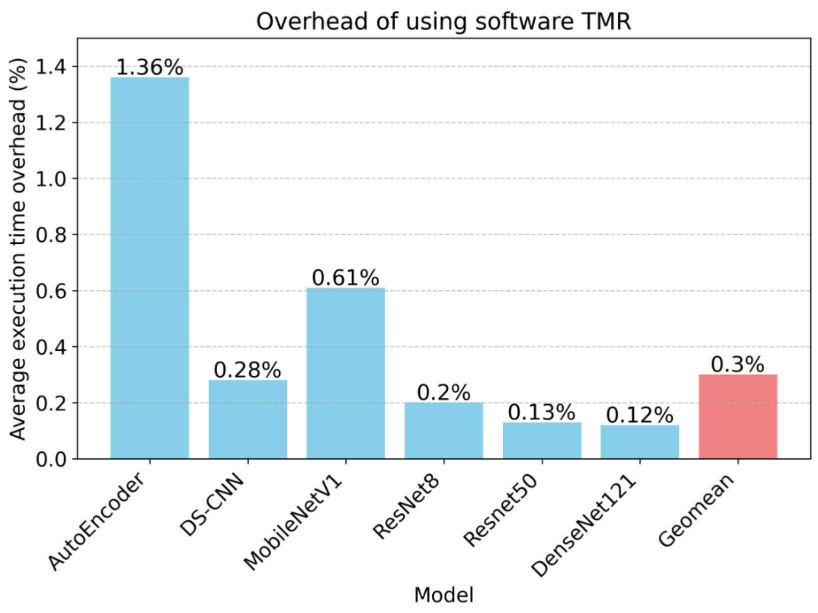


Figure 7.1: Average performance overhead of implementing TMR in terms of the execution times

Table 7.2: The percentage of N-bit-flip runs (out of 1000) that cause a 10% accuracy drop with and without using TMR.

Model	% Without TMR		% With TMR
	$N = 16$	$N = 32$	$N = 32$
AutoEncoder	91.5%	97.2%	0.0%
DS-CNN	99.8%	100.0%	0.4%
MobileNetV1	66.4%	89.8%	0.0%
ResNet-8	99.8%	100.0%	0.4%
ResNet-50	19.8%	52.8%	0.0%
DenseNet121	88.9%	99.0%	0.7%

the attack can be successful even using TMR when bits in different copies that correspond to the same bit in the same parameter are flippable (at least two out of three) **and** the model happens to be vulnerable to flipping the bit at that location. Evidently, this only happens in the highly vulnerable models (DS-CNN, ResNet-8, and DenseNet121), in which managing to corrupt at least two copies that map to the same bit is highly likely to damage the model significantly. Also, ResNet-50, which we found to be inherently less vulnerable than other models, is further fortified, bringing the percentage of successful attacks from 52.8% to 0.0% against a 32-bit-flip BFA.

To further gauge the effectiveness of our software mechanism, we estimate the likelihood of a *successful attack*. We define such an attack as an adversary's ability to bypass TMR by corrupting at least one *vulnerable bit* whose corruption results in a relative accuracy degradation of $\delta = 10\%$ or more. We assume that a random bit in memory is flipped with a probability p. To capture the variability in Rowhammer-flippable bits, we show the attack success probability (ASP) for different values of p in Figure 9.1. As p increases, the most vulnerable models (those with the most vulnerable bits in the high-precision regions) become more susceptible to the attack. Nevertheless, even in the most vulnerable models and at notably high p values, the ASP remains minimal given that the attacker is usually constrained in time and the number of bit-flips they can induce.

Chapter 8

Related Work

Due to their versatility, deep learning models have been increasingly employed in various applications. Many applications are highly safety-critical, and accuracy fluctuations can lead to severe consequences [79, 14, 2]. Hence, multiple works have discussed the attack and defence sides of DNNs.

Rakin et al. [62] demonstrates one of the first effective bit-flip attacks against quantized models using a bit-search algorithm. However, they assume that any bit in DNN model parameters is flippable, which is unrealistic. Yao et al. [84] demonstrate one of the first practical BFAs against DNNs using Rowhammer, which also uses a bit-search algorithm. ZeBRA [56] is another BFA vector that does not require access to training or test data and relies on synthetic data samples. The three mentioned works require white-box access to the model architecture, all the parameters, and a batch of testing data. Such requirements may not always be available. To our knowledge, our proposed attack against quantized DNNs is the first BFA that assumes a semi-black box threat model wherein the attacker only knows the model architecture without having access to model parameters or test data. T-BFA [63] is a recent *targeted* BFA that makes a ResNet-18 misclassify *all* inputs of class x to class y in the test set. Ghavami et al. [18] demonstrate another targeted BFA that maintains the model's accuracy for legitimate inputs while effectively neutralizing adversarial defences, causing even a protected model to misclassify adversarial inputs. Our work concentrates on untargeted BFA and defences against them.

Various works have introduced defences specific to BFAs and increased the reliability against natural bit faults. NeuroPots [47] injects known vulnerable parameters that a vulnerable bit-search algorithm would detect and protects those parameters by performing a checksum. Using NeuroPots can incur a significant performance overhead and can require model re-training. Aegis [75] defends against targeted BFA by including extra *internal classifiers (ICs)*, one of which can be randomly chosen as an early exit path during inference. Aegis requires model re-training to include the ICs and only defends against targeted BFAs. Forget and Rewire (FaR) [54] is a defence specific for vision transformers that neutralizes gradient-based attacks such as DeepHammer [84] by rewiring important parameters to neurons with no influence on the model output, i.e., neurons with activation values close to zero for a batch of the training set, at inference time. Zhou et al. [87] also show that vision transformers that use knowledge distillation [73] are more resilient against bit-flip attacks.

Ranger [8] corrects random bit faults by checking the range of activation values to avoid error propagation through the network. Instead of using global, layer-wise activation ranges, FitAct [19] uses trainable activation functions to provide per-neuron ranges. However, activation-limiting approaches have two problems. 1) They introduce hyperparameters that require per-model profiling and calibration, and 2) they can wrongfully clip activations for out-of-distribution inputs. Wasim et al. [77] primarily focuses on random faults in *computation* and requires model retraining and augmentation. Our work focuses on defending models against *data* adversarial BFA and can be deployed without modifying the model. Mahmoud et al. [50] selectively protect the most vulnerable feature maps in CNNs by computing the likelihood that an error in computation would propagate to the output. However, it can incur a performance overhead of 48% and only works for CNNs. In contrast, we propose a lightweight solution to protect sensitive parameters for quantized models without incurring a significant overhead. Furthermore, we show that our solution works for various network architectures.

Defences against random, transient hardware faults (to which the two previous works belong) assume a non-adversarial environment. In contrast, we assume the presence of an adversary launching BFA, intentionally degrading model accuracy with a high success rate.

Chapter 9

Limitations and Future Work

This thesis analyzed attacks and proposed defences for FP32 and quantized DNNs. This section notes some limitations of this work and describes potential future opportunities.

9.1 Range Constraints for FP32 Model Parameters

Our solution for BFA against FP32 models is grounded in our empirical observation that the parameter ranges are generally confined to smaller values. For models with parameters that span a larger range, some bit-flips that drastically alter a parameter to an extreme value can go undetected by the defence. This limitation can be effectively mitigated through interventions at either the model architecture level or via post-training adjustments. On the architecture level, several methods exist to mitigate this issue without compromising model accuracy during training. These include regularization [49, 41, 30] and early stopping [60], which help, either directly or indirectly, maintain weights at smaller values. Additionally, as we have discussed in section 5, the utilization of per-tensor ranges for comparison can constrain vulnerable regions only to the tensors with larger parameter ranges. Those tensors could then undergo additional protection strategies akin to those implemented for quantized models.

9.2 Algorithm-based BFA Against Quantized Models

We demonstrate an effective semi-black box BFA against quantized models and propose and analyze a defence strategy using redundancy. Prior works generally assume a white-box threat model in which the attacker has access to not only the model architecture but also all parameter values and a subset of the training or testing sets, which may not be realistic for all systems. In this case, the attacker can run sophisticated algorithms to select the most vulnerable bits in the low-precision parameters (weights) that can induce cumulative accuracy degradation. Our defence mechanism does not protect against such attacks as our threat model assumes the attacker does not know the exact values of model parameters. However, since the same algorithms can be run before deploying the model, the most vulnerable bit

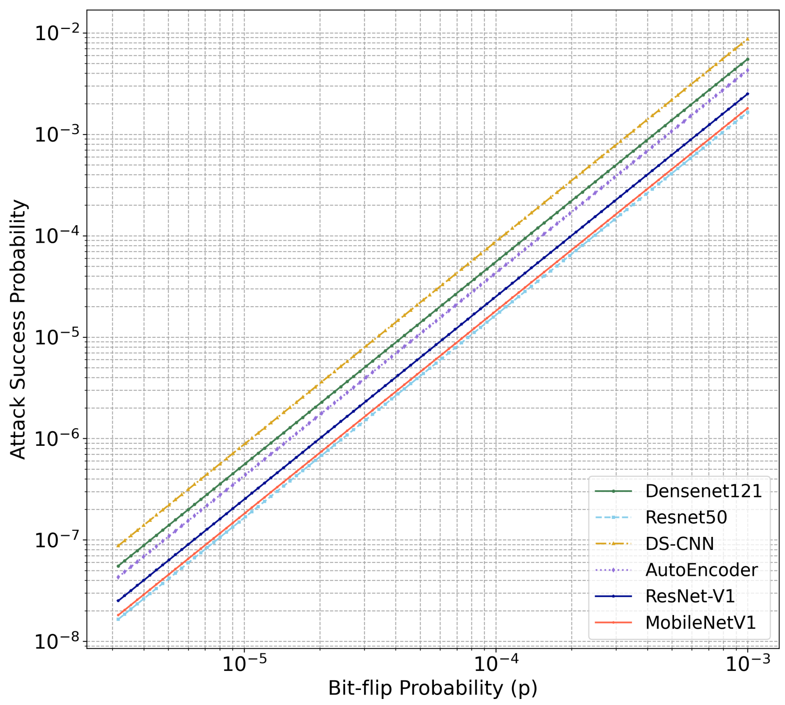


Figure 9.1: Attack success probability for different values of bit-flip probability p.

chains can be identified and hardened using an approach similar to what we described for high-precision parameters. A defence against white-box attacks using similar principles to our solution is a topic for future work.

9.3 Circumventing TMR

Depending on the distribution of the flippable bit cells, the attacker can flip bits in different copies that map to the same parameter and circumvent the TMR voting mechanism. Our randomized analysis (assuming vulnerable bits are uniformly distributed) shows a negligible attack success rate. However, OS/inference software may map the data to more vulnerable regions in physical memory, increasing the attack success rate. A possible solution would be implementing TMR for vulnerable parameters in hardware, which can randomize the locations of the replicas in memory, making it harder for the attacker to identify bit locations that correspond to the same bit in the three replicas. However, such hardware solutions might incur significant complexity for edge devices. Another solution would be keeping extra copies depending on the security requirement. For instance, keeping five replicas reduces ASP from 10^{-3} to approximately 10^{-6} for $p = 10^{-3}$ for ResNet-50 while increasing the storage overhead from 1.25% to 2.1% of the model size.

Chapter 10

Conclusion

In this work, we analyzed the risk of bit-flips against FP32 models. We proposed a simple solution based on the fact that most vulnerability comes from flipping the most significant bits in the exponent field. We highlighted a vulnerability in quantized DNNs, which exploits the fact that important, low-volume parameters are kept at a higher precision. We also show that this vulnerability requires no sensitivity studies or complex algorithms to find the most damaging bit-flips from the attacker's perspective. We exploit this vulnerability in an attack to make many DNN models completely malfunction (i.e., reach the level of a random guesser) by flipping a few bits without requiring open access to the model parameters and training or testing data. For instance, we managed to degrade a DenseNet121 model from 85% accuracy to a random guesser accuracy level (10%) using ten bit-flips on average. We also demonstrate the risk of such an attack against LLMs by testing it against GPT2, which produces meaningless outputs using only 20 bit-flips with a high probability. We propose and implement low-overhead TMR in the TensorFlow Lite Micro, guaranteeing strong reliability in aggressive attack scenarios without incurring significant storage and performance overheads.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Rakshit Agrawal, Jack W. Stokes, Karthik Selvaraj, and Mady Marinescu. Attention in recurrent neural networks for ransomware detection. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3222–3226, 2019.

[3] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[4] 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. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, page 743–755, New York, NY, USA, 2016. Association for Computing Machinery.

[5] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. MLPerf tiny benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[6] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 515–532, Santa Clara, CA, August 2019. USENIX Association.

[7] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti,

Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, aug 2011.

[8] Z. Chen, G. Li, and K. Pattabiraman. A low-cost fault corrector for deep neural networks through range restriction. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–13, Los Alamitos, CA, USA, jun 2021. IEEE Computer Society.

[9] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018, 2019.

[10] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the effectiveness of ECC memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 55–71, 2019.

[11] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. Tensorflow Lite Micro: Embedded machine learning for TinyML systems. *Proceedings of Machine Learning and Systems*, 3:800–811, 2021.

[12] Timothy J Dell. A white paper on the benefits of chipkill-correct ECC for PC server main memory. *IBM Microelectronics division*, 11(1-23):5–7, 1997.

[13] Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. Davit: Dual attention vision transformers. In *European Conference on Computer Vision*, pages 74–92. Springer, 2022.

[14] Jose Dolz, Karthik Gopinath, Jing Yuan, Herve Lombaert, Christian Desrosiers, and Ismail Ben Ayed. Hyperdense-net: a hyper-densely connected cnn for multi-modal image segmentation. *IEEE transactions on medical imaging*, 38(5):1116–1126, 2018.

[15] Apostolos Fournaris, Lidia Pocero Fraile, and Odysseas Koufopavlou. Exploiting hardware vulnerabilities to attack embedded system devices: a survey of potent microarchitectural attacks. *Electronics*, 6(3):52, Jul 2017.

[16] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Grand pwning unit: Accelerating microarchitectural attacks with the GPU. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 195–210, 2018.

[17] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Trrespass: Exploiting the many sides of target row refresh. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 747–762, 2020.

[18] Behnam Ghavami, Seyd Movi, Zhenman Fang, and Lesley Shannon. Stealthy attack on algorithmic-protected dnns via smart bit flipping. In *2022 23rd International Symposium on Quality Electronic Design (ISQED)*, pages 1–7, 2022.

[19] Behnam Ghavami, Mani Sadati, Zhenman Fang, and Lesley Shannon. Fitact: Error resilient deep neural networks via fine-grained post-trainable activation functions. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1239–1244, 2022.

[20] 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, Los Alamitos, CA, USA, may 2018. IEEE Computer Society.

[21] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. Prefetch side-channel attacks: Bypassing smap and kernel aslr. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 368–379, New York, NY, USA, 2016. Association for Computing Machinery.

[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] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraș. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 497–514, 2019.

[24] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[25] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 385–399, New York, NY, USA, 2020. Association for Computing Machinery.

[26] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[27] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

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

[29] Intel. Intel e7500 chipset datasheet. 2002.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[31] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[32] JEDEC. DDR4 SDRAM registered DIMM design specification. pages 4.20.28–5, 2014.

[33] Ruimin Ke, Yifan Zhuang, Ziyuan Pu, and Yinhai Wang. A smart, efficient, and reliable parking surveillance system with edge artificial intelligence on iot devices. *IEEE Transactions on Intelligent Transportation Systems*, 22(8):4962–4974, 2020.

[34] Jeremie S. Kim, Minesh Patel, A. Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. Revisiting rowhammer: an experimental analysis of modern dram devices and mitigation techniques. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ISCA '20, page 638–651. IEEE Press, 2020.

[35] 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 *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372, 2014.

[36] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering from the next row over. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3807–3824, Boston, MA, August 2022. USENIX Association.

[37] Yuma Koizumi, Shoichiro Saito, Hisashi Uematsu, Noboru Harada, and Keisuke Imoto. Toyadmos: A dataset of miniature-machine operating sounds for anomalous sound detection, 2019.

[38] Alexander Kozlov, Ivan Lazarevich, Vasily Shamporov, Nikolay Lyalyushkin, and Yury Gorbachev. Neural network compression framework for fast model inference. *arXiv preprint arXiv:2002.08679*, 2020.

[39] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018.

[40] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 (Canadian institute for advanced research). 2009.

[41] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. NIPS'91, page 950–957, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[42] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. Rambleed: Reading bits in memory without accessing them. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 695–711, 2020.

[43] Eojin Lee, Ingab Kang, Sukhan Lee, G. Edward Suh, and Jung Ho Ahn. Twice: Preventing row-hammering by exploiting time window counters. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 385–396, 2019.

[44] Yehao Li, Yingwei Pan, Ting Yao, and Tao Mei. Comprehending and ordering semantics for image captioning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 17990–17999, 2022.

[45] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[46] Moritz Lipp, Michael Schwarz, Lukas Raab, Lukas Lamster, Misiker Tadesse Aga, Clémentine Maurice, and Daniel Gruss. Nethammer: Inducing Rowhammer faults through network requests. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 710–719, 2020.

[47] Qi Liu, Jieming Yin, Wujie Wen, Chengmo Yang, and Shi Sha. NeuroPots: Realtime proactive defense against Bit-Flip attacks in neural networks. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6347–6364, Anaheim, CA, August 2023. USENIX Association.

[48] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 131–138, 2017.

[49] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[50] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Christopher W. Fletcher, Sarita V. Adve, Charbel Sakr, Naresh Shanbhag, Pavlo Molchanov, Michael B. Sullivan, Timothy Tsai, and Stephen W. Keckler. Optimizing selective protection for cnn resilience. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 127–138, 2021.

[51] Radu Marculescu, Diana Marculescu, and Umit Ogras. Edge ai: Systems design and ml for iot data analytics. KDD '20, page 3565–3566, New York, NY, USA, 2020. Association for Computing Machinery.

[52] Assaf Michaely, Carolina Parada, Frank Zhang, Gabor Simko, and Petar Aleksic. Keyword spotting for Google assistant using contextual speech recognition. In *ASRU 2017*, 2017.

[53] Szymon Migacz. 8-bit Inference with TensorRT. Technical report, NVIDIA, 2018.

[54] Najmeh Nazari, Hosein Mohammadi Makrani, Chongzhou Fang, Hossein Sayadi, Setareh Rafatirad, Khaled N. Khasawneh, and Houman Homayoun. Forget and rewire: Enhancing the resilience of transformer-based models against bit-flip attacks. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024.

[55] Ataberk Olgun, Majd Osseiran, Abdullah Giray Yağlıkcı, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. An experimental analysis of rowhammer in hbm2 dram chips, 2023.

[56] Dahoon Park, Kon-Woo Kwon, Sunghoon Im, and Jaeha Kung. Zebra: Precisely destroying neural networks with zero-data based repeated bit flip attack. *arXiv preprint arXiv:2111.01080*, 2021.

[57] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae W. Lee. Graphene: Strong yet lightweight row hammer protection. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13, 2020.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[59] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM addressing for Cross-CPU attacks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 565–581, Austin, TX, August 2016. USENIX Association.

[60] Lutz Prechelt. *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[61] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[62] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1211–1220, 2019.

[63] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-bfa: Targeted bit-flip adversarial weight attack. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7928–7939, 2021.

[64] 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, August 2016. USENIX Association.

[65] Nadav Rotem, Jordan Fix, Saleem Abdulrasool, Garret Catron, Summer Deng, Roman Dzhabarov, Nick Gibson, James Hegeman, Meghan Lele, Roman Levenstein, et al. Glow: Graph lowering compiler techniques for neural networks. *arXiv preprint arXiv:1805.00907*, 2018.

[66] Raj Sachdev. Towards security and privacy for edge ai in iot/ioe based digital marketing environments. In *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 341–346, 2020.

[67] Anish Saxena, Gururaj Saileshwar, Prashant J. Nair, and Moinuddin Qureshi. Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 108–123, 2022.

[68] Mark Seaborn. How physical addresses map to rows and banks in dram. 2015.

[69] Mark Seaborn and Thomas Dullien. Exploiting the dram rowhammer bug to gain kernel privileges. 2015.

[70] Kirill A. Shutemov. pagemap: Do not leak physical addresses to non-privileged userspace. 2015.

[71] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. CCS '17, page 587–601, New York, NY, USA, 2017. Association for Computing Machinery.

[72] 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, July 2018. USENIX Association.

[73] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 18–24 Jul 2021.

[74] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine 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*, CCS '16, page 1675–1689, New York, NY, USA, 2016. Association for Computing Machinery.

[75] Jialai Wang, Ziyuan Zhang, Meiqi Wang, Han Qiu, Tianwei Zhang, Qi Li, Zongpeng Li, Tao Wei, and Chao Zhang. Aegis: Mitigating targeted bit-flip attacks against deep neural networks. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2329–2346, Anaheim, CA, August 2023. USENIX Association.

[76] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[77] Syed Talal Wasim, Kabila Haile Soboka, Abdulrahman Mahmoud, Salman H Khan, David Brooks, and Gu-Yeon Wei. Hardware resilience properties of text-guided image classifiers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 67886–67899. Curran Associates, Inc., 2023.

[78] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. ACSAC '18, page 393–406, New York, NY, USA, 2018. Association for Computing Machinery.

[79] Li-Hua Wen and Kang-Hyun Jo. Fast and accurate 3d object detection for lidar-camera-based autonomous vehicles using one shared voxel-based backbone. *IEEE Access*, 9:22080–22089, 2021.

[80] J. Woo, G. Saileshwar, and P. J. Nair. Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 374–389, Los Alamitos, CA, USA, mar 2023. IEEE Computer Society.

[81] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. Open dnn box by power side-channel attack. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(11):2717–2721, 2020.

[82] 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, August 2016. USENIX Association.

[83] Ren Yang, Mai Xu, and Zulin Wang. Decoder-side hevc quality enhancement with scalable convolutional neural network. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pages 817–822, 2017.

[84] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1463–1480. USENIX Association, August 2020.

[85] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom. Pthammer: Cross-user-kernel-boundary rowhammer through implicit accesses. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 28–41, Los Alamitos, CA, USA, oct 2020. IEEE Computer Society.

[86] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *ArXiv*, abs/1606.06160, 2016.

[87] Xuan Zhou, Souvik Kundu, and Peter Anthony Beerel. What makes vision transformers robust towards bit-flip attack? In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024.