WeightSentry: Real-Time Bit-Flip Protection for Deep Neural Networks on GPUs

Mahmoud Abumandour
Simon Fraser University
Burnaby, British Columbia, Canada
mahmoud_abumandour@sfu.ca

Srinija Ramichetty
George Washington University
Washington, DC, USA
sramichetty9@gwmail.gwu.edu

Guru Venkataramani
George Washington University
Washington, DC, USA
guruv@gwu.edu

Alaa R. Alameldeen
Simon Fraser University
Burnaby, British Columbia, Canada
alaa@cs.sfu.ca

Abstract

Bit-flip attacks (BFAs) pose a significant threat to safety-critical applications that rely on deep neural networks (DNNs), enabling an attacker to deplete model accuracy by flipping a small number of bits. Existing defenses protect models by modifying the training process or the model architecture, or by applying hash- or checksum-based integrity checks. These approaches suffer from high performance overhead, limited correction capability, pristine model accuracy degradation, or poor scalability to GPU inference. We propose WeightSentry, a lightweight runtime defense that enforces per-weight fine-grained range checks without modifying the model architecture or requiring data-dependent calibration. WeightSentry detects and corrects bit flips by iteratively flipping the high-order exponent bits in the floating point representation until the weight falls within its valid range. WeightSentry is optimized for weight-level parallelism on GPUs, retains up to 100% of the pristine model accuracy under white-box BFA, and incurs only 1.1% runtime performance overhead. For large language models (LLMs), it imposes less than 0.00004% storage overhead while limiting the increase in perplexity score under attack from over 3,400,000% increase to less than 1.9%. Compared to two integrity-based defenses, HASHTAG and RADAR, it achieves up to 350× lower runtime and 2400× smaller storage overhead on GPUs.

CCS Concepts

• **Security and privacy → Hardware attacks and countermeasures**.

Keywords

Bit-flip attacks, Deep neural networks, GPU security, Fault injection attacks, Rowhammer

ACM Reference Format:
Mahmoud Abumandour, Srinija Ramichetty, Guru Venkataramani, and Alaa R. Alameldeen. 2025. WeightSentry: Real-Time Bit-Flip Protection for Deep Neural Networks on GPUs. In *Hardware and Architectural Support for Security and Privacy 2025 (HASP 2025), October 19, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3768725.3768734

1 Introduction

Deep Neural Networks (DNNs) power various critical applications, such as medical diagnosis and treatment [3, 23], autonomous vehicles [11, 22], and cybersecurity systems [7, 31]. While the usage of DNNs has achieved tremendous progress in those applications, various recent studies have shown that they are highly vulnerable to fault injection attacks [8, 24, 25, 33] that can deplete the intelligence of a DNN and convert it to a random guesser by corrupting a small number of model parameters. These attacks, also known as Bit-Flip Attacks (BFA), exploit the Rowhammer vulnerability [13] to flip bits in the model data.

In response, several defenses have been proposed to mitigate bit-flip attacks, which modify the model architecture [16], the training process [18], the model data representation [6, 27], or add runtime integrity checks [1, 12, 14, 15]. However, these methods often incur high performance and storage overheads, require model retraining or per-model calibration, and are specific to one type of task (e.g., classification). Furthermore, almost all defenses focus on quantized models and do not evaluate floating-point (FP) models [12, 15, 18], claiming that only the most significant bit (MSB) of the exponent is vulnerable to BFA, which is simple to protect. We demonstrate that in both convolutional neural networks (CNNs) and large language models (LLMs), progressively flipping other high-order bits can slowly cause significant accuracy or perplexity degradation.

Defenses using integrity checks can successfully detect up to 100% of bit flips with 0% false positive rate. However, they are designed for CPU inference and do not scale well to GPU-based inference due to their sequential nature. Recent work has demonstrated Rowhammer attacks on GPU memory [17], which exposes modern inference platforms, such as those running LLMs, to these attacks with no efficient defense options. Furthermore, models often undergo post-deployment optimization and fine-tuning to adapt to the distribution shift between the training dataset and real-world data [28]. Even though such optimization may introduce minimal changes to model parameters, it means that golden hashes or signatures must be recalculated to reflect the modified weights, which can incur a significant performance overhead.

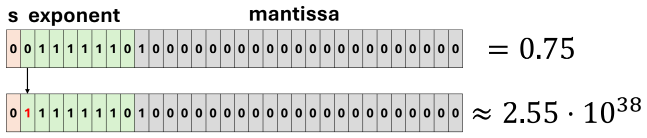


Figure 1: The effect of flipping the most significant bit (MSB) of the exponent in a 32-bit IEEE-754 floating-point value (0.75) increases its magnitude to $\sim 2.55 \times 10^{38}$**.**

To address these limitations, we propose WeightSentry, a lightweight defense that enforces fine-grained runtime anomaly detection of model weights. Unlike prior approaches that rely on offline calibration, model retraining, or cryptographic signatures, WeightSentry leverages statistical properties already present in the trained model to define per-channel (for CNNs) or per-tensor (for LLMs) weight bounds. We exploit the vulnerability in FP models where flipping the high-order exponent bits from 0 to 1 can increase the magnitude of parameters to extreme values and corrupt all downstream computations [8], as shown in Table 1. During inference, weights are continuously validated against those bounds, and any out-of-range values are automatically clamped by iteratively flipping exponent bits until the value is in the valid range. WeightSentry requires no model retraining or calibration, introduces negligible performance and storage overheads, and protects models used for classification and open-ended language tasks. Specifically, WeightSentry exploits parameter-level parallelism and achieves a high GPU utilization, incurring 1.1% performance overhead. WeightSentry outperforms two representative defenses, HASHTAG [12] and RADAR [15], by 350×, and 31×, respectively, and reduces storage overhead by up to 32× and 2400×.

In summary, this paper makes the following contributions.

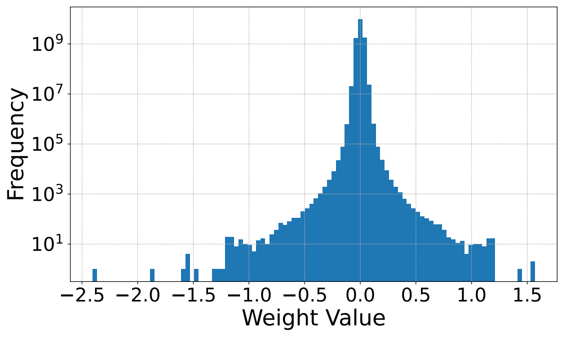
- We show that FP weights can be stealthily attacked using gradient-based BFAs without targeting the exponent's MSB, degrading both CNNs and LLMs.
- We propose WeightSentry, a lightweight, GPU-optimized defense against BFAs on FP DNNs that requires no model retraining, calibration, or architectural changes.
- WeightSentry detects and corrects weights in real-time by iteratively flipping exponent bits, achieving 100% accuracy retention in CNNs and reducing LLM perplexity increases from more than 3,400,000% to less than 1.9%. Compared to HASHTAG and RADAR, WeightSentry achieves up to 350× lower runtime overhead and 2400× smaller metadata footprint when running on GPU platforms.

2 Background

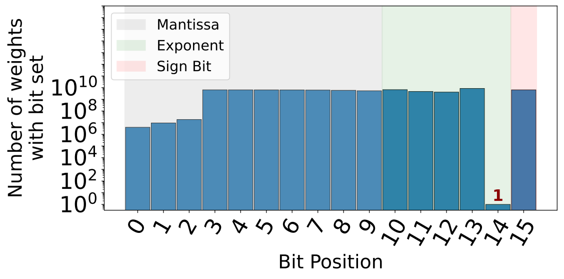
In this section, we present an overview of the BFAs, the Rowhammer vulnerability used to trigger them, and existing defenses that protect deep neural networks against BFAs.

2.1 Bit-Flip Attacks

Bit-Flip Attacks (BFA) against DNNs fall into two categories: targeted and untargeted attacks. A targeted BFA is a stealthy attack that aims to cause misclassifications for *specific input samples* while preserving the model's accuracy on all other inputs [26]. In contrast, an untargeted BFA's objective is to degrade the overall model accuracy as much as possible using a small number of bit flips [8, 25, 33].



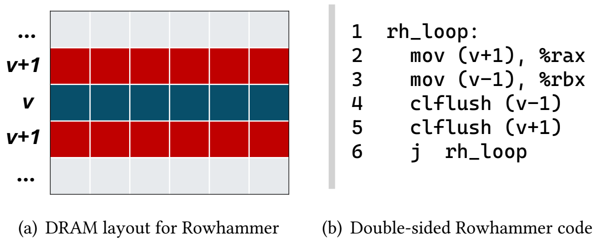
(a) Weight Value Distribution



(b) Bit Distribution

Figure 2: 2(a) shows the parameter value distribution for the Llama 3.2 13B model using the half-precision FP16 representation. It shows that all weight values are centered around zero. 2(b) shows the bit distribution for the same model. A bar at Bit Position = i **corresponds to the number of parameters with the** i**th bit equal to 1 (the left-most element in 2(a), which corresponds to roughly** -2.5**). Only a single parameter out of 13 billion parameters (**$7.7 \cdot 10^{-9}$**%) has the most significant exponent bit set to 1.**

Models stored in FP representation are highly susceptible to BFA. The parameters of such models are stored using the IEEE-754 format, where flipping the most significant bits in the exponent field can push values to extreme magnitudes, severely affecting the model's behavior. As illustrated in Figure 1, flipping the most significant exponent bit of a typical value (0.75) can transform it into an extremely large number ($2.55 \cdot 10^{38}$). Hong et al. [8] demonstrate that a single bit flip in the exponent of more than 40% of model parameters can cause substantial accuracy degradation across various models. More recent works have introduced targeted gradient-based BFAs [25, 33], which progressively select bits for flipping by maximizing the loss gradient with respect to weight bits. Given a weight tensor $\mathcal{W}$ and model loss function $\mathcal{L}$, the attacker computes the gradient $\nabla_{b_i}\mathcal{L}$ of the loss with respect to each bit b_i. The attacker selects the bit b^*, which maximizes the gradient:



```
rh_loop:
  mov (v+1), %rax
  mov (v-1), %rbx
  clflush (v-1)
  clflush (v+1)
  j  rh_loop
```

(a) DRAM layout for Rowhammer (b) Double-sided Rowhammer code

Figure 3: Double-sided Rowhammer demonstration. v is the victim row, and the attacker repeatedly accesses adjacent rows v+1 and v-1 to induce bit-flips in v.

$$b^* = \text{argmax}_{b_i \in \mathcal{W}} \nabla_{b_i} \mathcal{L}$$

Running this algorithm on FP models, it almost always discovers exponent MSB bits, since flipping them causes the largest increase in loss. However, such flips are easily detected by tracking the legitimate parameters that have this bit set in the original model. For example, as shown in Figure 2(b), only a single parameter has that bit set to 1 in the Llama-3.2 13B model, which is the only value with magnitude greater than 2.0, as shown in Figure 2(a). A defense could zero out all other occurrences during inference. However, an attacker, in response, can use stealthier variants of gradient-based attacks, which avoid flipping MSBs and instead target less conspicuous bits and degrade the model more gradually over time.

2.2 Flipping Bits using Rowhammer

Dynamic Random Access Memory (DRAM) is vulnerable to device disturbance errors, where repeated access to a specific DRAM row accelerates charge leakage in adjacent rows, potentially causing neighboring cells to lose their state, and the stored bit is flipped. The Rowhammer attack [13] exploits disturbance errors to flip crucial bits in victim data. As shown in Figure 3, the attacker performs Double-sided Rowhammer by rapidly accessing *aggressor rows* **v+1** and **v-1**, which can cause bit flips in the targeted victim row **v** due to accelerated charge leakage. To cause a bit to flip, the minimum number of row accesses is known as the *Rowhammer Threshold (T_{RH})*. Due to aggressive DRAM scaling, T_{RH} has been decreasing, making DRAM chips more vulnerable. Rowhammer has been exploited to break the security of various applications, including launching BFA against DNNs [8, 24, 33]. Recent work has also demonstrated Rowhammer faults in modern GPUs, showing that device-level disturbance is not limited to DRAM modules attached to CPUs, and can be exploited in GPU memory systems as well [17].

2.3 Existing BFA Defenses

Many defenses have been developed to protect DNNs against fault injection and BFA at different levels of abstraction in the deep learning (DL) stack, including modifications to the model architecture [16], the training process [18], data representation [6, 27], or the use of runtime mechanisms [1, 15].

At the architecture level, shadow models are co-deployed with the main model; mismatches in predictions trigger a fault, which

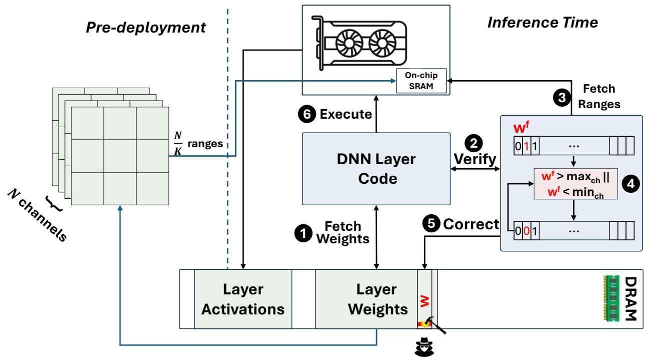


Figure 4: A high-level overview of the pre- and post-deployment operations of WeightSentry.

results in inference recomputation or reloads data from the trusted storage [16]. However, this approach is dependent on the accuracy of the shadow model and can incur a significant performance overhead if an accurate (large and complex) shadow model is chosen.

At training time, Liu et al. [18] modify the optimization process to deliberately create vulnerable honeypot bits that are monitored using a checksum. Rakin et al. [6, 27] address BFA by binarizing weights, which reduces fault sensitivity but can lead to a substantial loss in accuracy.

Runtime integrity defenses [12, 15] periodically verify parameter blocks against stored signatures. Since the signatures are stored in DRAM, they are also susceptible to bit flips, allowing adversaries to bypass detection by flipping bits in both data and signatures. Error Correction Codes (ECC) can detect and correct a limited number of errors, but recent studies show Rowhammer attacks can still succeed even on ECC-protected DRAM chips [2]. These defenses are typically implemented using hash- [12] or checksum-based mechanisms [15, 18], which are inherently sequential and poorly suited for parallel execution. As a result, they incur substantial performance penalties when deployed on modern GPU inference platforms. Activation anomaly detectors [1, 14], pre-profile models to compute activation ranges on the training set. During inference, all intermediate outputs outside of those ranges are clipped. This approach fails for out-of-distribution data samples often encountered post-deployment [28].

Importantly, many of the existing defenses have two common problems: they only work for integer-only quantized models and do not generalize to FP models; and they work only for classification tasks but are unsuitable for open-ended tasks such as those handled by LLMs.

3 WeightSentry Design

Figure 4 presents a high-level overview of WeightSentry, our proposed defense against BFA on FP models. The core insight driving WeightSentry is based on the observation that nearly all impactful BFA vulnerabilities in FP models originate from flipping the most significant bit (MSB) of the exponent field from 0 to 1, which results in extreme parameter changes. These anomalous weights propagate through the network, affecting all activation computation, which leads to a cascading error accumulation. Even a single bit flip in

Table 1: Distribution of top-ranked (largest delta in the loss) bit positions selected by a gradient-based BFA. Exponent bits are chosen in over 99% of the rounds.

Model	Exp. MSB (bit 30)	Exp. other (bits 23–29)	Mantissa+Sign (0–22, 31)
ResNet-18	10224 (96.3%)	289 (2.7%)	105 (1%)
MobileNetV2	1530 (96.5%)	54 (3.4%)	1 (0.06%)

exponent bits can corrupt computations in all subsequent layers, regardless of the layer type and following activation function [8].

Figure 2(b) shows the bit distribution of the Llama 3.2 13B model. Out of more than 13 billion parameters, only a single value has the exponent's MSB set to 1. This confirms that any such occurrence during inference is likely the result of a bit-flip. More generally, large out-of-range weights caused by bit-flips are almost always produced by corruptions in the exponent field. We validate this vulnerability pattern using a gradient-based BFA, similar to the method proposed in [25] (see Table 1). For both ResNet-18 and MobileNetV2, the vast majority of top-ranked bits selected for flipping are in the exponent field (approximately 99% of the selected bits in both models). Based on this observation, WeightSentry recovers the correct values by toggling each bit in the exponent field in order of significance, and selects the first bit-toggle that restores the value within the valid range. If no such correction is found, e.g., due to multiple flips or mantissa corruption, the weight is clipped to the nearest range boundary. Rather than invalidating entire blocks of parameters as in prior defenses [12, 15], WeightSentry targets individual outlier weights and attempts to restore them in-place. This approach avoids unnecessary corrections of bit flips that do not cause an observable perturbation, and eliminates the need to immediately reload large parameter blocks of data from trusted storage.

3.1 Pre-Deployment Phase

Prior to model deployment, WeightSentry computes fine-grained, static per-layer weight bounds from the trained model. For convolutional layers, these are computed on each output channel independently (i.e., per-kernel) if the kernel is larger than 3×3. Convolutional layers with smaller kernels are grouped into kernel groups of K weights each, as shown in Figure 4. The parameter K balances detection accuracy and storage overhead. Smaller K values provide finer-grained bounds, as shown in Figure 5, and improve detection sensitivity to small parameter changes.

For dense and attention layers, such as those used in fully connected networks (FC), Transformers, and LLMs, the bounds are computed per tensor to balance resolution with storage cost. For instance, a single range is used for each of the query (Q), key (K), and value (V) projection matrices in attention layers. These bounds are computed post-training from the minimum and maximum values per target unit (channel or tensor). No additional profiling or data-dependent calibration runs are required. Once computed, these bounds are stored in on-chip SRAM, commonly available and accessible in modern commodity GPUs and FPGAs. For instance, in CUDA, `__shared__` memory refers to the on-chip SRAM physically located within each Streaming Multiprocessor (SM) of the GPU. It offers lower latency and higher bandwidth than global DRAM, but has limited capacity (typically 48–228 KB per SM for current GPUs) and is shared among all the threads in a thread block. Unlike DRAM, SRAM is not vulnerable to Rowhammer, allowing the system to rely on the integrity of the range metadata even with the presence of an adversary.

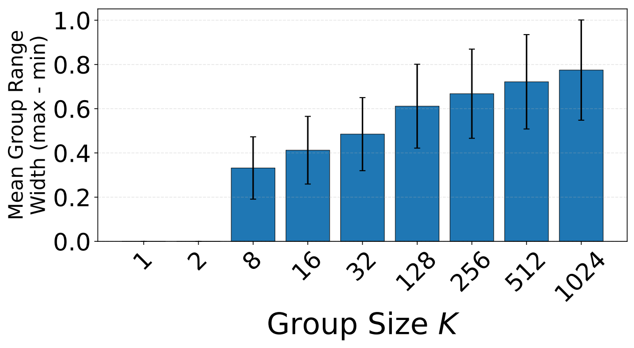


Figure 5: Average group parameter range ($\max - \min$) for different group sizes K on 1×1 kernels for MobileNetV2. For $K \leq 2$, the range is effectively zero, as individual weights can be stored directly with equivalent or half the cost of storing range bounds, which eliminates all possible bit-flips. As K increases, the average range width grows, allowing for more subtle bit-flips to go undetected.

3.2 Post-Deployment Phase

During inference, weights are loaded from memory and passed through the WeightSentry checker before being used in computation. The checker compares each weight to its corresponding range. If the weight is out of range, WeightSentry attempts to correct it by iteratively toggling each bit in the exponent field, starting from the MSB. If toggling a single bit produces a value within the unit range, the corrected weight is used and committed to memory. If no such correction produces an in-range value, the weight is clamped to the nearest boundary of the valid range. This ensures that even in the rare cases of non-exponent flips, only a tolerable error (within parameter range) is propagated through the network. Since *online learning* can modify the weight values after deployment, we periodically recompute all parameter ranges to reflect these updates. The WeightSentry correction process is detailed in Algorithm 1.

3.3 Threat Model

We assume a strong white-box model where the attacker has full knowledge of the model architecture and parameters, and shares the execution platform with the victim, allowing them to flip bits in the victim's data stored in DRAM using Rowhammer. The attacker's goal is to degrade model accuracy by corrupting a small number of high-impact weights. The attacker is capable of identifying the most vulnerable parameters by analyzing model weights [26, 33]. Since the effectiveness of Rowhammer depends on the specific DRAM chip vulnerability, we also consider a more constrained white-box model in which the attacker can accurately locate the model weights in memory but can only induce random bit flips. This represents the

Algorithm 1: WeightSentry Runtime Correction

```
Input: Weight $w$, min bound $w_{\min}$, max bound $w_{\max}$
Output: Corrected weight $\hat{w}$
if update_period() then
    $(w_{\min}, w_{\max}) \leftarrow$ compute_bounds()
if $w_{min} \le w \le w_{max}$ then
    return $w$
for bit $b$ in exponent bits (MSB to LSB) do
    $w' \leftarrow$ toggle_bit($w, b$)
    if $w_{min} \le w' \le w_{max}$ then
        return $w'$
if $-w \in [w_{min}, w_{max}]$ then
    return $-w$
return clip($w, w_{\min}, w_{\max}$)
```

situation in which the DRAM used to store weights exhibits limited or inconsistent vulnerability to precise Rowhammer targeting.

4 Evaluation

4.1 Experimental Setup

We evaluate WeightSentry on tasks from two domains: vision and natural language. For vision, we use four CNNs: ResNet-18 [5], ResNet-50, MobileNetV2 [29], and DenseNet121 [9]. These models are trained on the CIFAR-10 dataset, which contains 60,000 32×32 colored images divided evenly over 10 classes. For natural language, we evaluate four language models suitable for edge inference: LLaMA-3.2-3B [4], Qwen2.5-VL-3B [32], Phi-4-Mini [19], and SmolLM3-3B [10]. These models are evaluated using the perplexity metric on a 500-sentence subset of the WikiText-101 dataset. Perplexity measures how well a language model predicts a sample, with lower values indicating better predictive performance. For a model assigning probability $P(t_i)$ to each token t_i in a sequence of N tokens, perplexity is defined as follows.

$$\text{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(t_i)\right)$$

4.1.1 Training and Precision. All CNNs are trained for 100 epochs with a learning rate of 0.001, followed by 20 epochs at 0.0001, using the Adam optimizer. Inference is performed in `float32` precision. LLMs are evaluated using pretrained checkpoints in `float16` precision, using their default deployment configurations.

4.1.2 Hardware and Implementation. All experiments and model training are run on an NVIDIA RTX 4090 GPU with 24 GB of GDDR6X memory. WeightSentry is implemented in CUDA 12.6 and integrated in PyTorch 2.7.

4.1.3 BFA Evaluation. We test WeightSentry against two attack variants: flipping random bits in weights and a gradient-ranking attack, similar to that proposed in [25], but adapted for FP representation. We use a 200-image subset from the CIFAR-10 test dataset for testing CNNs, and measure the pristine model accuracy, the accuracy under attack, and the accuracy when the model is defended by WeightSentry. For LLMs, we calculate the perplexity score over a 500-sentence sample from the WikiText-101 dataset [30].

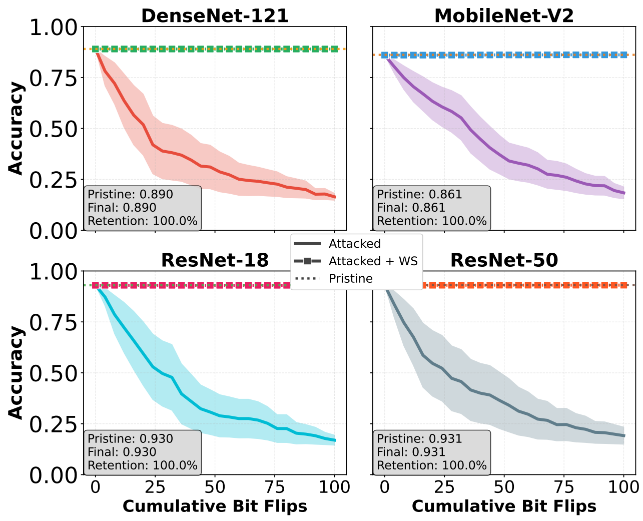


Figure 6: Average accuracy degradation due to a 100-bit Random BFA with and without WeightSentry. WeightSentry retains 100% of the pristine accuracy for all tested models.

4.1.4 Comparison Baselines. We port two prior defenses, HASHTAG [12] and RADAR [15], to run on GPUs and compare them against WeightSentry. HASHTAG detects bit-flips by computing the Pearson hash [21] over weight tensors to generate signatures before deployment, and verifying the parameters used for inference against the signatures for integrity at runtime. RADAR detects bit-flips using a checksum-based 2-bit signature over weight groups. On detecting an integrity violation during inference, RADAR replaces all the elements in the corrupted groups with zeros. For all defenses, we assume that parameters are fine-tuned post-deployment (e.g., due to online learning) and trigger metadata recomputation every eight inference invocations. For WeightSentry, we assume a kernel grouping granularity of $K = 128$ unless otherwise specified.

4.2 Mitigation of Bit-Flip Attacks

Existing works often rely on the assumption that an attacker will only flip a few, predictable bits [8, 15] (usually the MSB), and that model accuracy can be retained by protecting those bits. WeightSentry aims to not only catch those predictable BFAs, but also stealthier attacks in which the attacker aims to gradually degrade the model over time. We examine the effect of bit-flips under two models: one where the attacker is constrained in the DRAM susceptibility to flips, and another where they can flip any subset of bits in the model data.

4.2.1 Random BFA. A fully-random BFA does not incur a significant degradation in model accuracy, as most of the low-order mantissa bits do not significantly change parameter values. To model a moderately powerful attacker, we flip random bits in the top two bytes of model weights. With just 100 bit-flips, unprotected models gradually collapse from an accuracy greater than 85% to less than 20%. We progressively flip bits and measure the accuracy degradation. As shown in Figure 6, WeightSentry retains 100% of the pristine model accuracy.

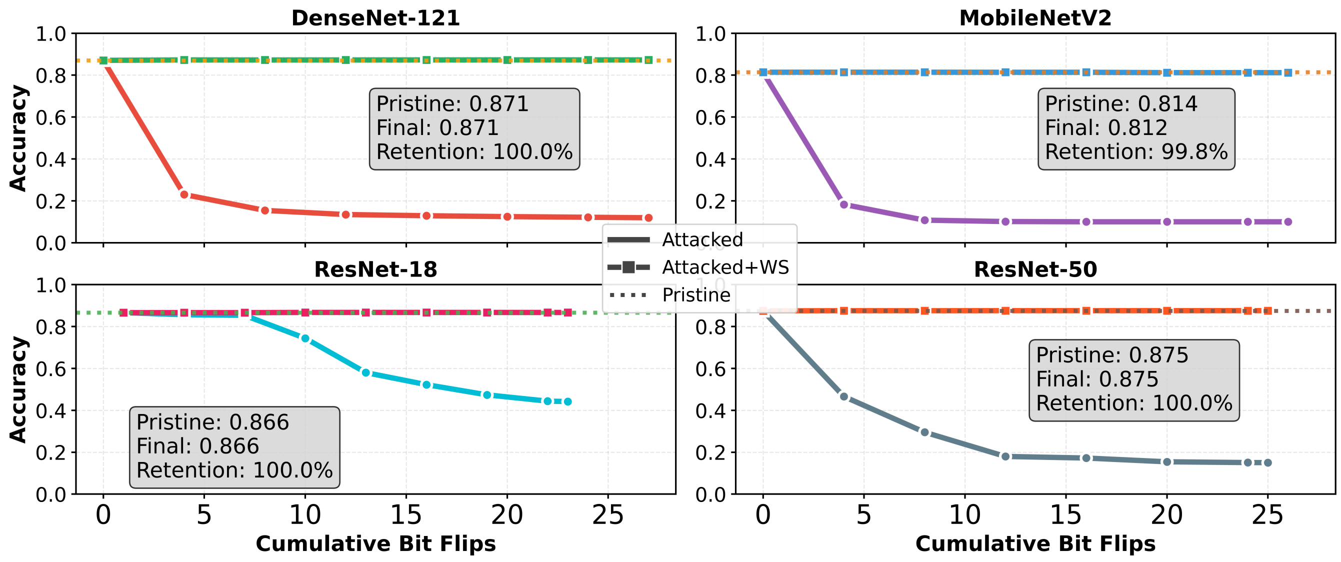


Figure 7: 25-bit Gradient-Based S-BFA results against CNNs. WeightSentry retains 100% of three models and only loses 0.2% for MobileNetV2 due to its kernel granularity.

4.2.2 Gradient-Based Bit-Flips. Having access to weights and gradient information, an attack can identify the bits that can induce the largest increase in loss. However, due to the nature of FP numbers, an automatic gradient-ranking algorithm returns exponent MSBs with a very high rate, as shown in Table 1. Such an attack is effective and can degrade a model into a random guesser using a single bit-flip, but it is easily detectable, and WeightSentry is guaranteed to accurately correct all such cases.

We therefore focus on a stealthier attack (S-BFA), in which the attacker picks top-ranked bits, but avoids those that incur a large, easily detectable, delta in the loss. This constraint prevents flips that produce extreme outliers (e.g., exponent's MSB), and instead targets bit flips that cause subtle changes that cause accumulative degradation in model accuracy. Hence, the objective of S-BFA becomes as follows, where τ is a predefined loss delta threshold. In our evaluation, we set $\tau = 10^3$, which eliminates all conspicuous exponent MSB flips:

$$b^* = \arg\max_{b_i \in \mathcal{W}} \Delta\mathcal{L}_{b_i} \quad \text{subject to} \quad \Delta\mathcal{L}_{b_i} < \tau$$

This variant is slower to degrade the model, but is harder to detect as the parameter perturbations it produces are more subtle. We report accuracies with and without WeightSentry in Figure 7 and Figure 8 for CNNs and LLMs, respectively. For CNNs, S-BFA degrades three of the four models to random guessers with roughly 25 bit flips. WeightSentry correctly identifies all bits S-BFA selects in DenseNet-121, ResNet-18, and ResNet-50, and retains 100% of the pristine model accuracy. For MobileNetV2, WeightSentry detects all high-impact flips except for a single, low-order mantissa bit, shown in Table 1, that changes a weight value from 0.007481 to 0.006992. The corrupted value remains within the valid range and goes undetected by WeightSentry. However, the corresponding bit is the only mantissa bit found in the top 100 parameter bits in gradient ranking, and such flips are generally extremely rare. Even though their impact on accuracy is minor, such rare cases can be handled by augmenting the defense with targeted redundancy or by using finer-grained kernel grouping granularity K, as discussed in Section 4.5.

Table 2: Attacked LLM response (S-BFA) to the prompt "What is the meaning of life?" with and without WeightSentry. Llama-3.2 outputs repeated tokens while Phi-4-Mini outputs a non-English response.

Model	Attacked Output	Defended Output
Llama-3.2-3B	The 2019 2019 2019 The...	It's a question that has puzzled philosophers ...
Phi-4-Mini	Drachenkönig wohnt auf einem hohen Hügdynamisch	It has puzzled humanity since time immemorial.

For LLMs, the perplexity score is more fragile. Unlike classification accuracy, perplexity is affected even by small parameter perturbations. When the models are undefended, S-BFA increases the perplexity by several orders of magnitude, up to more than 3,400,000% for Phi-4-Mini. With WeightSentry, the same attack *improves* the perplexity score of Phi-4-Mini marginally by 0.3%. In the worst case for WeightSentry, S-BFA degrades (i.e., increases) the perplexity score by less than 1.9% for Llama-3.2-3B compared to a perplexity score increase of more than 2,000,000% in the unprotected Llama-3.2-3B model. Table 2 shows example differences in model behavior under a 25-bit S-BFA. The attacked LLama model repeatedly generates the same tokens, while Phi produces an irrelevant[1], non-English response. In contrast, the defended models

[1]Phi's response literally translates to "Dragon King lives on a high hilldynamic" from German.

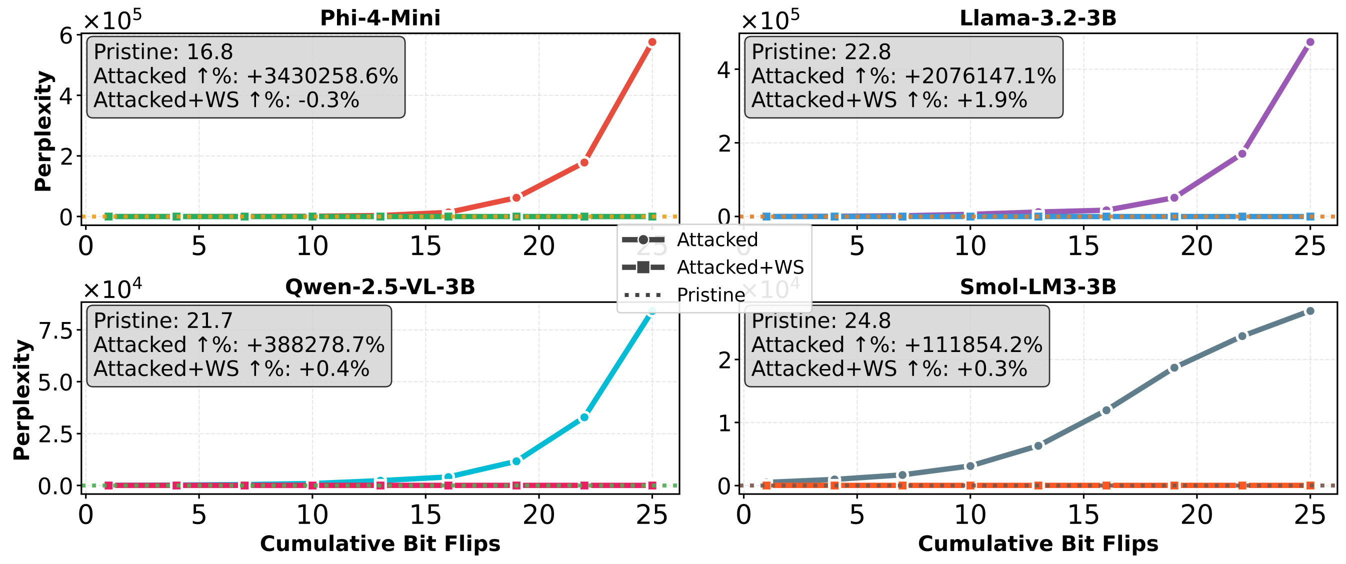


Figure 8: 25-bit Gradient-Based S-BFA results against LLMs. For example, the perplexity score increases by 1.9% for Llama-3.2-3B with WeightSentry (worst case), whereas the unprotected model's perplexity score increases by more than 2,000,000% after 25 bit flips. Worst-case perplexity increase for the unprotected model is more than 3,400,000% in the Phi-4-Mini model, while WeightSentry reduces its perplexity by 0.3%.

generate fluent, relevant completions, similar to those generated by the pristine models.

4.3 Storage Overhead

WeightSentry is designed to keep its metadata compact to reside entirely in on-chip SRAM. This serves two purposes: SRAM is resistant to Rowhammer attacks, ensuring its integrity, and SRAM is also faster to access during execution. WeightSentry decouples its storage cost from parameter count and maintains only two values per weight unit. Table 3 compares storage overhead across models for two WeightSentry variants: WS128 and WS256, which correspond to $K = 128$ and $K = 256$, respectively. We assume that all three defenses are used to protect the entire model, and do not selectively pick the most sensitive layers. For CNNs, both WeightSentry variants use less space than both HASHTAG and RADAR, except in MobileNetV2. The exception is due to the large number of small convolutional kernels in the MobileNetV2 architecture, increasing the number of groups G, each treated as a weight unit. Even in this case, the total storage is only 1.17 KB. WeightSentry scales significantly better in larger models by maintaining a single range per weight unit, and, unlike RADAR, it does not depend on the number of parameters. While the metadata overhead for both WeightSentry and HASHTAG depends on the number of protected units, WeightSentry's per-layer cost is only 8 bytes per weight unit compared to HASHTAG's fixed 256-byte table. Across all tested LLMs, WeightSentry requires less than 7KB of metadata, which is up to 2400× smaller than RADAR and 32× smaller than HASHTAG.

Table 3: Storage overhead (in kilobytes) comparison across models. Both variants of WeightSentry WS128 and WS256, with $K = 128$ and $K = 256$, scale better for larger models.

Model	HT	RD	WS128	WS256
ResNet-50	40.41	22.43	**4.47**	2.73
ResNet-18	15.56	10.66	**2.64**	1.54
DenseNet121	91.36	6.63	**3.81**	3.32
MobileNetV2	37.65	**2.13**	5.55	3.27
Llama-3.2-3B	63.75	3063.92	**1.98**	**1.98**
Qwen2.5-3B	206.80	3580.69	**6.44**	**6.44**
Phi-4-Mini	48.69	3658.32	**1.52**	**1.52**
SmolLM3-3B	81.82	2932.64	**2.55**	**2.55**

4.4 Runtime Performance

We port RADAR and HASHTAG by re-implementing them in CUDA to run on the GPU and evaluate their runtime overhead. We report the normalized performance degradation (latency) and throughput. We integrate the protections into PyTorch and benchmark inference on different batch sizes. WeightSentry scales more efficiently with increasing batch size and approaches the unprotected baseline performance and throughput. Figure 9 reports the average results for running 1000 batches on the ResNet-18 model, assuming the defenses are applied to all layers. For batch size 64, WeightSentry achieves 8722 images/sec (vs. 8849 images/second for the unprotected baseline) with only 1.5% overhead, while RADAR incurs 34.8% overhead and HASHTAG over 280% overhead.

4.4.1 Selective Protection. In all previous results, we assumed that protections are applied to all layers in the model. Prior works have

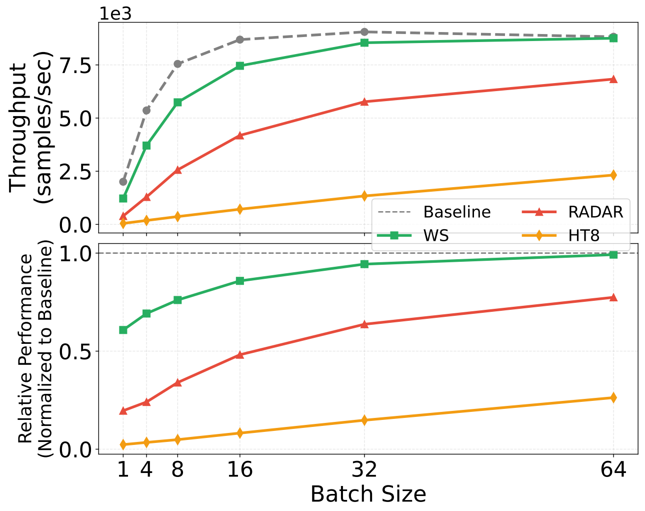


Figure 9: The throughput (top) and relative performance (bottom) of HASHTAG (HT8), RADAR, and WeightSentry (WS) applied to all ResNet-18 layers. For large batch sizes (≥ 32), WeightSentry approaches the performance and throughput of the unprotected baseline.

shown that vulnerability can concentrate in a few layers, and protecting them can capture almost all impactful bit-flips and reduce the storage overhead [8, 12]. We adopt per-layer sensitivity-based selective protection, applying defenses only to the three most vulnerable layers identified via a gradient-based BFA. To avoid repeatedly selecting the same bit-flip sequence, we run the attack on multiple input batches to introduce variance in the computed gradient values and randomize vulnerable bit selection. Then, we select the layers that are most frequently targeted across all runs. Figure 10 shows that WeightSentry outperforms other defenses, approaching baseline performance. It incurs only 2.8% overhead at batch size 16 (vs. 26% for RADAR and 275% for HASHTAG) and 1.1% at batch size 64.

4.5 Varying Protection Granularity

Some architectures, such as MobileNetV2, use a large number of 1×1 kernels, which consist of a single weight and are used to perform dimensionality transformations. To address this, we perform *kernel grouping*. Kernels with size ≤ 3×3 are grouped into units of K kernel weights, and a single range is computed per group. This creates a storage-accuracy tradeoff: coarser ranges reduce storage but detect subtle parameter changes less accurately, and finer ranges improve detection accuracy at the cost of higher metadata overhead. We run a 50-bit S-BFA against MobileNetV2 and DenseNet121 and show the accuracy after attack for different group sizes K in Figure 11.

For $K \leq 2$, we can achieve perfect protection by storing a trusted, redundant copy of the 1×1 kernels directly in SRAM. This allows us to retain 100% of the pristine model accuracy and detect all bit-flips for such kernels. While this increases the storage requirement for MobileNetV2 to 95.23KB from 5.55KB for $K = 128$, all metadata can still fit in a single streaming multiprocessor's (SM) shared SRAM

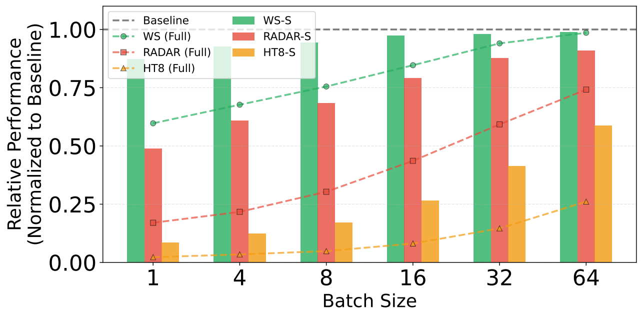


Figure 10: The overhead for applying protection selectively to the top 3 layers for WeightSentry (WS-S), RADAR (RADAR-S), and HASHTAG (HT8-S). WS-S approaches baseline performance for lower batch sizes and outperforms RADAR-S and HT8-S for all batch sizes.

in modern commercial GPUs such as the NVIDIA RTX 4090, which provides up to 99 KB of usable shared SRAM per SM [20].

As K increases, the ranges become coarser, and more subtle bit-flips go undetected. At $K = 1024$, six bit flips go undetected in MobileNetV2, and the accuracy goes down substantially from 81.4% to 71.9%. For DenseNet121, only a single bit flip is missed at $K = 256$ and $K = 1024$, causing a negligible accuracy degradation from 87.07% to 87.04%. Results show that moderate kernel grouping ($K = 128$) offers an effective balance between storage and protection accuracy for model architectures with fine-grained convolutions.

5 Conclusion

We present WeightSentry, a lightweight, GPU-optimized defense against bit-flip attacks on deep neural networks represented in floating-point (FP) format. Unlike prior defenses that require model retraining, integrity metadata, or task-specific calibration, WeightSentry enforces localized range checks derived from the post-training model parameter statistics. We demonstrate that FP models are susceptible to stealthy gradient-based BFA by constraining the attack to pick bits other than the exponent's MSB, which degrades the model gradually by introducing many small perturbations. WeightSentry detects and corrects such out-of-bounds weights during inference time with minimal performance and storage overhead. Specifically, it retains up to 100% of pristine model accuracy under two strong, white-box attack scenarios while incurring only 1.1% runtime overhead. It significantly outperforms existing integrity-based defenses when implemented in the GPU, achieving up to 350× less performance overhead and 2400× smaller metadata footprint.

Acknowledgments

This work was supported, in part, by the Natural Sciences and Engineering Research Council of Canada (NSERC) under RGPIN-2021-03729, an SFU Faculty Recruitment Grant, the Office of Naval Research under Grant N00014-24-1-2046, and gifts from Intel Corporation and Ampere Computing.

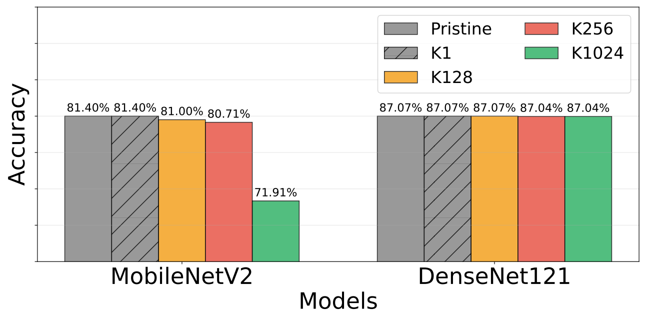


Figure 11: Model accuracy under a 50-bit S-BFA for different kernel group sizes K. Smaller K improves detection and accuracy retention, at the cost of higher storage overhead. MobileNetV2 starts to suffer from accuracy degradation due to undetected bit-flips, while DenseNet121 remains unaffected for coarser range granularities.

References

[1] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. 2021. 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)*. 1–13. doi:10.1109/DSN48987.2021.00018

[2] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. 2019. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*. 55–71. doi:10.1109/SP.2019.00089

[3] Jose Dolz, Karthik Gopinath, Jing Yuan, Herve Lombaert, Christian Desrosiers, and Ismail Ben Ayed. 2018. HyperDense-Net: a hyper-densely connected CNN for multi-modal image segmentation. *IEEE transactions on medical imaging* 38, 5 (2018), 1116–1126. https://doi.org/10.1109/TMI.2018.2878669

[4] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, and Amy Yang *et al.* 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[6] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. 2020. Defending and Harnessing the Bit-Flip Based Adversarial Weight Attack. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 14083–14091. doi:10.1109/CVPR42600.2020.01410

[7] Elike Hodo, Xavier J. A. Bellekens, Andrew W. Hamilton, Christos Tachtatzis, and Robert C. Atkinson. 2017. Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey. *CoRR* abs/1701.02145 (2017). arXiv:1701.02145 http://arxiv.org/abs/1701.02145

[8] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraș. 2019. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium (USENIX Security 19)*. 497–514. doi:10.5555/3361338.3361373

[9] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely Connected Convolutional Networks. *CoRR* abs/1608.06993 (2016). arXiv:1608.06993 http://arxiv.org/abs/1608.06993

[10] HuggingFaceTB. 2024. SmolLM3-3B. https://huggingface.co/HuggingFaceTB/SmolLM3-3B

[11] Jyh-Jing Hwang, Runsheng Xu, Hubert Lin, Wei-Chih Hung, Jingwei Ji, Kristy Choi, Di Huang, Tong He, Paul Covington, Benjamin Sapp, Yin Zhou, James Guo, Dragomir Anguelov, and Mingxing Tan. 2024. EMMA: End-to-End Multimodal Model for Autonomous Driving. arXiv:2410.23262 [cs.CV] https://arxiv.org/abs/2410.23262

[12] Mojan Javaheripi and Farinaz Koushanfar. 2021. HASHTAG: Hash Signatures for Online Detection of Fault-Injection Attacks on Deep Neural Networks. *CoRR* abs/2111.01932 (2021). arXiv:2111.01932 https://arxiv.org/abs/2111.01932

[13] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. 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)*. 361–372. doi:10.1109/ISCA.2014.6853210

[14] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. 2017. Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications. In *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[15] Jingtao Li, Adnan Siraj Rakin, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. 2021. RADAR: Run-time Adversarial Weight Attack Detection and Accuracy Recovery. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 790–795. doi:10.23919/date51398.2021.9474113

[16] Yu Li, Min Li, Bo Luo, Ye Tian, and Qiang Xu. 2020. DeepDyve: Dynamic Verification for Deep Neural Networks. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 101–112. doi:10.1145/3372297.3423338

[17] Chris S. Lin, Joyce Qu, and Gururaj Saileshwar. 2019. GPUHammer: Rowhammer Attacks on GPU Memories are Practical. In *28th USENIX Security Symposium (USENIX Security 19)*. 497–514. doi:10.5555/3361338.3361373

[18] Qi Liu, Jieming Yin, Wujie Wen, Chengmo Yang, and Shi Sha. 2023. NeuroPots: Realtime Proactive Defense against Bit-Flip Attacks in Neural Networks. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 6347–6364. https://www.usenix.org/conference/usenixsecurity23/presentation/liu-qi

[19] Microsoft, Abdelrahman Abouelenin, Atabak Ashfaq, Adam Atkinson, and Hany Awadalla *et al.* 2025. Phi-4-Mini Technical Report: Compact yet Powerful Multimodal Language Models via Mixture-of-LoRAs. arXiv:2503.01743 [cs.CL] https://arxiv.org/abs/2503.01743

[20] NVIDIA. 2023. NVIDIA ADA GPU ARCHITECTURE. https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf

[21] Peter K. Pearson. 1990. Fast hashing of variable-length text strings. *Commun. ACM* 33, 6 (June 1990), 677–680. doi:10.1145/78973.78978

[22] Rui Qian, Xin Lai, and Xirong Li. 2022. 3D Object Detection for Autonomous Driving: A Survey. *Pattern Recognition* 130 (2022), 108796. doi:10.1016/j.patcog.2022.108796

[23] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M. Dai, Nissan Hajaj, Peter J. Liu, Xiaobing Liu, Mimi Sun, Patrik Sundberg, Hector Yee, Kun Zhang, Gavin E. Duggan, Gerardo Flores, Michaela Hardt, Jamie Irvine, Quoc V. Le, Kurt Litsch, Jake Marcus, Alexander Mossin, Justin Tansuwan, De Wang, James Wexler, Jimbo Wilson, Dana Ludwig, Samuel L. Volchenboum, Katherine Chou, Michael Pearson, Srinivasan Madabushi, Nigam H. Shah, Atul J. Butte, Michael D. Howell, Claire Cui, Greg Corrado, and Jeff Dean. 2018. Scalable and accurate deep learning for electronic health records. *CoRR* abs/1801.07860 (2018). arXiv:1801.07860 http://arxiv.org/abs/1801.07860

[24] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. 2022. DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories. In *2022 IEEE Symposium on Security and Privacy (SP)*. 1157–1174. doi:10.1109/SP46214.2022.9833743

[25] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2019. Bit-Flip Attack: Crushing Neural Network withProgressive Bit Search. *CoRR* abs/1903.12269 (2019). arXiv:1903.12269 http://arxiv.org/abs/1903.12269

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

[27] Adnan Siraj Rakin, Li Yang, Jingtao Li, Fan Yao, Chaitali Chakrabarti, Yu Cao, Jae-sun Seo, and Deliang Fan. 2025. RA-BNN: Constructing a Robust & Accurate Binary Neural Network Using a Novel Network Growth Mechanism to Defend Against BFA. In *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*. 00219–00228. doi:10.1109/CCWC62904.2025.10903977

[28] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. 2019. Do ImageNet Classifiers Generalize to ImageNet? *CoRR* abs/1902.10811 (2019). arXiv:1902.10811 http://arxiv.org/abs/1902.10811

[29] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *CoRR* abs/1801.04381 (2018). arXiv:1801.04381 http://arxiv.org/abs/1801.04381

[30] 2016 Stephen Merity et al. [n. d.]. Wikitext-103. ([n. d.]). https://arxiv.org/abs/1609.07843

[31] Peilun Wu and Hui Guo. 2019. LuNet: A Deep Neural Network for Network Intrusion Detection. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. 617–624. doi:10.1109/SSCI44817.2019.9003126

[32] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, and Fei Huang *et al.* 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[33] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. 2020. DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1463–1480. https://www.usenix.org/conference/usenixsecurity20/presentation/yao