Forget and Rewire: Enhancing the Resilience of Transformer-based Models against Bit-Flip Attacks

Najmeh Nazari, Hosein Mohammadi Makrani, and Chongzhou Fang, *University of California, Davis;* Hossein Sayadi, *California State University, Long Beach;* Setareh Rafatirad, *University of California, Davis;* Khaled N. Khasawneh, *George Mason University;* Houman Homayoun, *University of California, Davis*

https://www.usenix.org/conference/usenixsecurity24/presentation/nazari

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

Forget and Rewire: Enhancing the Resilience of Transformer-based Models against Bit-Flip Attacks

Najmeh Nazari
University of California, Davis

Hosein Mohammadi Makrani
University of California, Davis

Chongzhou Fang
University of California, Davis

Hossein Sayadi
California State University, Long Beach

Setareh Rafatirad
University of California, Davis

Khaled N. Khasawneh
George Mason University

Houman Homayoun
University of California, Davis

Abstract

Bit-Flip Attacks (BFAs) involve adversaries manipulating a model's parameter bits to undermine its accuracy significantly. They typically target the most vulnerable parameters, causing maximal damage with minimal bit-flips. While BFAs' impact on Deep Neural Networks (DNNs) is well-studied, their effects on Large Language Models (LLMs) and Vision Transformers (ViTs) have not received the same attention. Inspired by "brain rewiring," we explore enhancing Transformers' resilience against such attacks. This potential lies in the unique architecture of transformer-based models, particularly their Linear layers. Our novel approach, called *Forget and Rewire* (FaR), strategically applies rewiring to Linear layers to obfuscate neuron connections. By redistributing tasks from critical to non-essential neurons, we reduce the model's sensitivity to specific parameters while preserving its core functionality. This strategy thwarts adversaries' attempts to identify and target crucial parameters using gradient-based algorithms. Our approach conceals pivotal parameters and enhances robustness against random attacks. Comprehensive evaluations across widely used datasets and Transformer frameworks show that the FaR mechanism significantly reduces BFA success rates by 1.4 to 4.2 times with minimal accuracy loss (less than 2%).

1 Introduction

LLMs [33] and ViTs [10] have transformed tasks in natural language processing and computer vision, demonstrating impressive capabilities in text understanding and generation, and image classification [4, 9, 11, 15]. However, these models are vulnerable to hardware-induced adversaries, such as Bit-Flip Attacks (BFAs) [7, 18]. BFAs exploit the hardware layer to introduce errors in the memory regions storing a model's weight parameters, compromising the model's integrity and performance. Techniques like DeepHammer [49] target specific bits within a DRAM page, altering sensitive weights and degrading model performance.

Despite advances in memory technology, new methods can still remotely influence memory content without direct physical access [26]. Defending against BFAs is complex due to the gradient-based Progressive Bit Search algorithm, which identifies the most sensitive bits within the model's weights. Therefore, understanding and mitigating these attacks is crucial for ensuring the robustness and reliability of these advanced models in real-world applications.

Although modern memory technologies have enhanced resilience to BFAs, researchers continue to uncover novel methodologies capable of influencing the content of memory remotely, without the necessity of direct physical access [26]. Mounting a defense against BFAs is inherently a complex task. This complexity arises from the BFA's utilization of a gradient-based Progressive Bit Search algorithm, which meticulously identifies and modifies bits within the model's most sensitive weights.

Inspired by the renowned neuropsychologist Donald Hebb, who coined the well-known phrase "Neurons that fire together, wire together" [29], we discover parallels in the adaptability and resilience of the human brain in the face of challenges. This popular statement underscores the brain's capacity for ongoing reorganization and rewiring, adapting to our activities, thoughts, and emotions in a dynamic process. Similarly, if certain thought patterns no longer serve our best interests, our brains have the ability to rewire to adopt alternative thoughts, showcasing a remarkable capacity for change.

In this study, we draw inspiration from the brain's adaptive mechanisms to address analogous vulnerabilities in transformer-based architectures [42]. Our proposition centers on the reallocation of redundant parameters in transformer-based models such as LLM and ViT, redirecting them to augment more crucial parameters. This strategy ensures the collective operation of lesser-used parameters with pivotal ones, enhancing the model's resilience against hardware-induced adversaries such as BFAs [35, 36], while maintaining accuracy with negligible loss.

Taking cues from the tactics employed by attackers, this work introduces a novel defense mechanism aiming to obfus-

cate the transformer-based models' most sensitive weights. The proposed methodology identifies the most and least sensitive parameters within the model, subsequently rewiring the least sensitive parameters to those deemed most crucial. This redistribution of task responsibility across the parameters effectively diminishes the sensitivity gradient, rendering gradient-based search algorithms less effective in identifying key parameters. Essentially, this approach aims to cloak these important parameters, ensuring that any bit-flip injections target less pivotal parameters.

We present a thorough exploration of the theoretical and practical aspects of BFAs, specifically, considering the real-world constraints of LLMs and ViTs. Building on these insights, we introduce a framework (*Forget and Rewire: FaR*) dedicated to pinpointing sensitive parameters and bolstering their resilience against adaptive attacks.

Our comprehensive evaluations reveal that our proposed approach, while minimally affecting model accuracy (-1.97% on ImageNet without the need for retraining), significantly obfuscates critical weights by up to 84% and 91% from potential expert and basic attackers, respectively. Concurrently, FaR compels Oracle and basic attackers to execute 1.6x up to 4x more bit-flips to achieve the same level of disruption (over 10% accuracy degradation) as they would in the absence of FaR enhancement. To achieve acceptable robustness against BFAs by obfuscating 15% of parameters per layer, FaR incurs up to 4.5% and 9.3% overhead in inference time and model storage, respectively.

It is worth noting that our proposed defense mechanism exhibits considerable compatibility, allowing it to synergize with other defensive strategies, such as NeuroPots [25] and Aegis [43]. This compatibility is attributed to FaR's primary focus on enhancing the overall robustness of models rather than solely on attack detection and recovery.

To the best of our knowledge, this is the first work that adopts a rewiring method to conceal important parameters of transformer-based models, improving their resilience against the challenging model-centric bit-flip attacks. These attacks pose a significant challenge with an attack surface potentially too extensive for existing solutions to comprehensively cover. Last but not least, the FaR library is open-sourced and publicly available to the community for development and evaluation[1]. We hope that our results could provide a new perspective for defending against emerging attacks in deep learning, fostering further in-depth research in this direction.

2 Background

In this section, we present a concise overview of the essential preliminary knowledge, especially on Transformers, floating points representation in modern hardware, and bit-flip attacks.

[1]Access at: http://tinyurl.com/farusenixsecurity

2.1 Transformer-based Models

The Transformer architecture forms the backbone of Large Language Models and Vision Transformers, playing a pivotal role in contemporary, cutting-edge pipelines for natural language processing and image classification [1]. In this section, we provide details of the Transformer architecture, underscoring the compatibility and effectiveness of our approach when applied to this class of models.

The Transformer model has two main parts [42]: the encoder and the decoder, as shown in Figure 1. The encoder takes an input and turns it into a representation (features). This helps the model understand the input. On the other hand, the decoder uses the representation from the encoder and some other information to create an output sequence. Either of these two blocks can be employed independently based on the nature of the task at hand:

- Encoder-only: Suitable for tasks that require an understanding of the input, such as text classification and named entity recognition (NER).
- Decoder-only: Suitable for generative tasks.
- Encoder-decoder: Suitable for generative tasks that require input, such as summarization or translation.

A key characteristic of Transformer models lies in their incorporation of specialized layers known as attention layers. These layers instruct the model to give particular emphasis to specific words within a sentence, effectively prioritizing them while de-emphasizing others when constructing the representation of each word. The attention mask can also be used in the encoder/decoder to prevent the model from paying attention to some special words.

Delving into the multi-head attention layer, It is comprised of several Linear modules, each with its own set of weights [42]. Within the attention layer, there are three essential parameters: Query, Key, and Value. These three distinct Linear layers, which share a similar structure, treat each word in the sequence as a vector. The encoded input representation is passed through all three parameters. This process results in an updated encoded representation that includes attention scores for each word.

Within the Transformer architecture, the Attention module carries out its computations in parallel multiple times. Each of these parallel computations is referred to as an Attention Head. The Attention module divides its Query, Key, and Value parameters into N separate parts, processing each of these splits independently through its own dedicated Head. After these individual Attention calculations, which share a similar structure, are completed, their results are combined to generate a final Attention score. This approach is known as Multihead attention, and it empowers the Transformer to capture a broader range of relationships and subtleties associated with each word, enhancing its overall capacity.

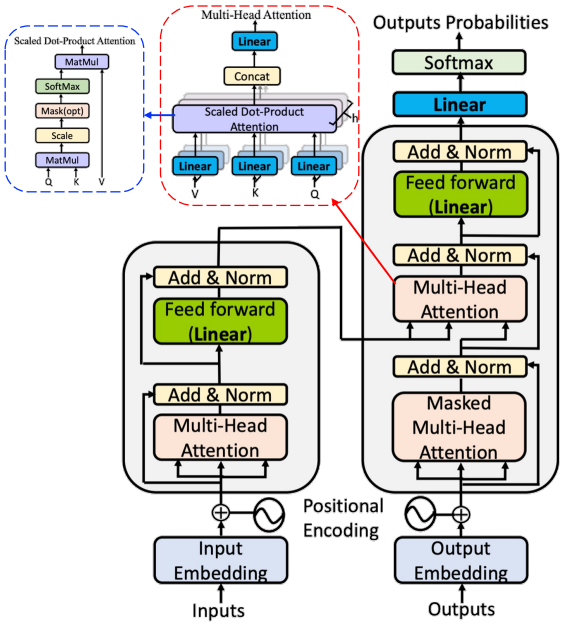

Figure 1: Architecture of Transformers.

2.2 IEEE 754 Floating Point Arithmetic

Weight parameters are typically represented using IEEE-754 32-bit single-precision floating-point numbers. This format exploits exponential notation but sacrifices precision for a broader range of possible values. For example, the value 0.34375 in exponential notation is expressed as 1.375×2^{-2}, where 1.375 is the mantissa, and -2 is the exponent. The IEEE754 single-precision floating-point format allocates 23 bits for the mantissa, 8 bits for the exponent, and one bit for the value's sign. What makes this format intriguing from an adversarial standpoint is the varying impact of different bits on the represented value. For instance, consider flipping the 20th bit in the mantissa, which results in a slight increase from 0.34375 to 0.359375, constituting a typically inconsequential perturbation. Conversely, flipping the highest exponent bit transforms the value into 1.375×2^{126}. Although both scenarios involve a single-bit manipulation, they produce two different outcomes.

2.3 Bit-Flip Attacks

The robustness, security, and safety of a modern computing system depend on the memory isolation enforced at the software and hardware level [12]. However, even in modern DRAM chips, memory isolation can be compromised by read disturbance. A notable instance of this is RowHammer [30], an extensively researched read-disturbance phenomenon. In RowHammer, the act of repetitively accessing and closing (hammering) a DRAM row numerous times results in bitflips

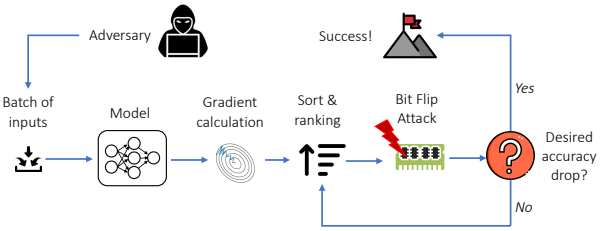

Figure 2: Bit-Flip Attack overview.

occurring in rows that are physically close by. Another example is RowPress [26], the most recent memory fault injection attack targeting DDR4-based systems that are hardened by RowHammer protection hardware. RowPress showed that even a user-level program can still breach memory isolation by maintaining an open state in a DRAM row for an extended duration, leading to disruptions in physically adjacent rows significant enough to induce bitflips.

The primary objective of bit-flip attacks is to use any practical memory fault injection attack to reduce the accuracy of a model while minimizing the number of bit-flips performed. A stealthy attacker's aim is to reclassify inputs originally belonging to source category p into a different target category q (where q is not equal to p) while ensuring that the remaining inputs maintain their original categories, thereby preserving their accuracy and ensuring the attack remains stealthy.

One of the most successful BFAs is the DeepHammer attack [49] which enhances BFAs by refining the algorithm used to search for vulnerable bits. DeepHammer also uses gradient-based Progressive Bit Search (PBS) to find vulnerable bits, as shown in Figure 2. In this attack, during the k^{th} iteration, DeepHammer first selects n vulnerable bits based on a gradient ranking of the model's parameters. Subsequently, DeepHammer individually flips each of these bits, generating a loss set denoted as L. This process is repeated for each layer, resulting in a total of $n \times l$ candidate bits and their corresponding loss set. DeepHammer identifies the most vulnerable bit as the one with the highest loss. Bit-flip is repetitively done until the desired outcome is achieved. While DeepHammer is limited to flipping just one bit per page, we are assuming that the adversary has the capability to flip any number of bits per DRAM page.

3 Motivation

3.1 Unveiling Security Challenges and Threats on LLMs and ViTs

The prevalence of transformer-based models such as LLMs and ViTs has surged dramatically, marking a significant shift in the landscape of machine learning. These sophisticated models, exemplified by transformer architectures, have found

extensive application across numerous domains, from machine translation [44] and content generation [50] to virtual assistants [21] and data analysis [13]. However, this proliferation has brought to the forefront a range of security concerns [5, 54]. The immense power of LLMs to generate coherent and contextually appropriate text can be harnessed by malicious actors for purposes such as misinformation dissemination, automated phishing, and even the creation of convincing fake content [53]. Additionally, LLMs and ViTs are not immune to adversarial attacks, where subtle input modifications can manipulate their output or compromise their decision-making processes [45]. As LLMs and ViTs become more deeply integrated into critical systems, the potential impact of these security vulnerabilities becomes more pronounced. Addressing these concerns is crucial to ensure that the benefits of transformer-based models are harnessed without undermining data integrity, and privacy in the digital landscape.

Security attacks on machine learning models can be categorized as follows: 1) evasion attacks [8], which involve perturbing inputs during testing to deceive the classification model; 2) poisoning attacks [47], aimed at manipulating training datasets to yield poorly-trained ML models; and 3) fault injection attacks [6], which modify ML parameters to alter the classifications of specific inputs towards target labels. Irrespective of the attack category, the overarching objective of adversarial attacks is to induce misclassifications in certain inputs, while maintaining a high model accuracy for others to remain stealthy.

3.2 Addressing the Research Gap

While numerous studies have explored the vulnerabilities of DNNs to fault attacks [32] or the susceptibility of LLMs and ViTs to evasion attacks [27], a significant gap in research remains unaddressed: the investigation of fault injection attacks specifically targeting transformer-based models. Fault attacks on DNNs have focused on exploiting vulnerabilities in their hardware implementations or manipulating their parameters to induce misclassifications. Similarly, evasion attacks on LLMs and ViTs have delved into generating adversarial inputs to deceive the models' decision-making processes. However, the distinct realm of fault injection attacks on LLMs [7] and ViTs [52], which involves introducing controlled memory corruptions or parameter perturbations to manipulate their output, has remained less explored. This uncharted territory presents a unique challenge, given the complex nature of transformer-based architecture and its diverse applications. Fault injection attacks on LLMs and ViTs could potentially result in unintended or biased text generation, or misclassification, undermine their coherence, or even enable subtle manipulations that are difficult to detect.

With the aim of uncovering the susceptibility of LLMs and ViTs to fault attacks, we embarked on an investigation. Our primary objective was to evaluate the resilience of these

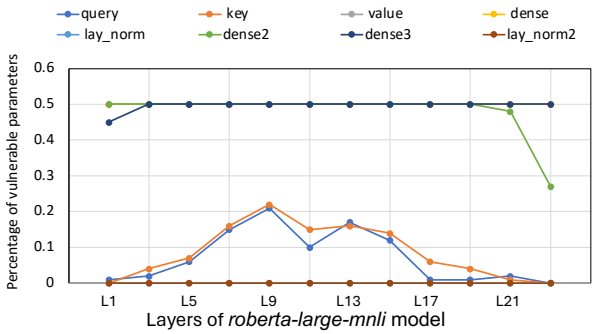

Figure 3: Vulnerability analysis of transformer layers and modules to single-BFA.

models by analyzing the manipulating individual model components and parameters through the introduction of bit-flips. We sought to understand how these induced faults could potentially compromise classification outputs, remaining imperceptible to human observers. In pursuit of this goal, we exposed transformer parameters to the latest BFAs.

Our analysis unveiled a critical observation: within LLMs and ViTs, certain parameters emerged as Achilles' heels. These select parameters represented singular points of failure within the model, where even a single bit-flip (SBF) could inflict severe performance degradation. This observation is supported by findings from Reference [7], which noted that single-bit-flips in certain *non-Wquery* and *Wkey* parameters could severely disrupt translation output, reducing BLEU scores to near zero. Single-bit-flip attacks, if successful, would lead to a complete model failure (e.g., accuracy drops to 0), often indicated by *"NaN"* values in PyTorch tensors. Since this behavior is easily detectable, it is not a favorite approach for attackers. It is important to emphasize that BFAs are more stealthy adversaries capable of flipping multiple bits per DRAM page to degrade the performance without being detected. Figure 3 shows an example of our analysis of SBF on *roberta-large-mnli* model from the hugging-face library. We randomly selected 100 parameters per layer of the transformer encoder and investigated if SBF can change the text classification output. The observation illustrates that dense3 and dense2 are the most vulnerable modules while normalization modules do not show any vulnerability.

Conversely, we also observed the existence of numerous neurons and parameters within the model that remained dormant, generating negligible activations without impacting model sensitivity. This observation led us to contemplate an innovative approach aimed at leveraging these non-essential parameters to enhance the model's robustness against BFAs.

3.3 Design Intuition

Consider a pre-trained ML model with parameter set P. In the presence of any input x, adversaries aim to find a universal set of weight bit-flips that result in misclassification, changing the correct label y to an incorrect label. In practice, adversaries

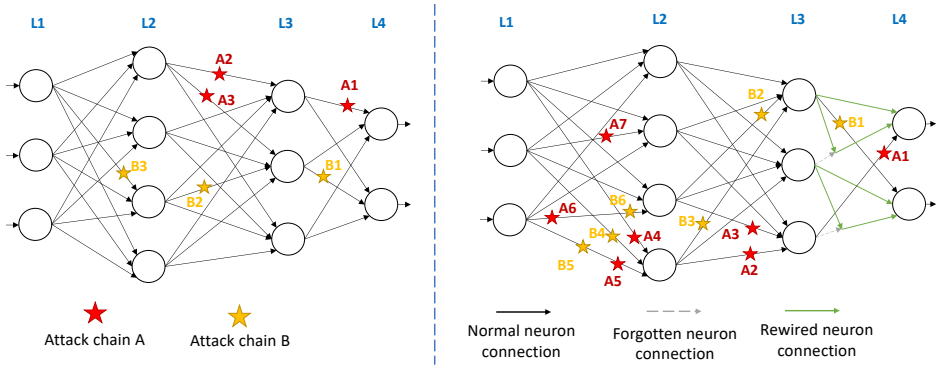

Figure 4: Comparison of the normal model and hardened model under BFAs. (a) The left diagram shows the position of bit-flips of 2 attack chains in a normal model. (b) The right diagram shows the position of bit-flips of 2 attack chains in a hardened model with FaR that demonstrates the increase in BFAs to achieve the same results.

are searching for a systematic misclassification of any valid input x by altering the fewest number of bits in the model parameter set P.

Drawing inspiration from concepts such as brain rewiring and guided by gradient-based search algorithms, we are driven to propose a methodology to reconfigure linear layer parameters. Our motivation is to diminish the model's sensitivity to a few critical parameters by redistributing their responsibilities to less crucial counterparts. In essence, our objective is to increase the model's resilience against BFAs by strategically rewiring its neural connections and optimizing its parameter utilization.

Figures 4 (a) and (b) illustrate the distribution of bit-flips in an attack chain within a linear model, both before and after rewiring. In the absence of rewiring (Figure 4(a)), the bit-flips within the attack chains can affect important weights across all layers. In Figure 4(b), we introduce a rewiring technique, where non-important parameters are interconnected with important ones in each layer. This serves to reduce the sensitivity of important parameters and conceal their significance. Consequently, if an attacker attempts to identify crucial parameters using gradient search, they will primarily discover less significant parameters that have minimal impact on the model's outputs. As a result, the attacker would need to flip a larger number of bits, rendering their efforts impractical.

The compatibility between our methodology and other detection and recovery techniques such as NeuroPots further enhances the overall resilience of the system, creating a more robust and comprehensive defense against a broader spectrum of potential threats and adversarial scenarios. NeuroPots operates by designating certain non-critical parameters as *bait* to detect attacks, and it checks them for any alteration to restore the original weights. In contrast, FaR is designed to enhance the robustness of the model against BFA. By redistributing critical parameter influence, FaR increases the number of bit-flips an attacker must execute to significantly degrade model performance. FaR focuses on preventing effective attacks by diluting parameter sensitivity.

Figure 5 illustrates our *Forget and Rewire* scheme. In Figure 5(a), we have a standard linear model where three neurons in layer $l-1$ are connected to one neuron in layer l. Suppose we perform sensitivity analysis on three parameters of W using a batch of input data. In this analysis, we find that X_2, the activation from neuron m_2, is predominantly close to 0, making W_2 the least important. Similarly, because X_1 is larger than X_3, the gradient of the output with respect to W_1 surpasses that of W_3. Furthermore, the output of neuron Y depends solely on $X_1W_1 + X_3W_3$.

In Figure 5(b), we showcase the layer following the application of our Forget and Rewire scheme. Since m_2 is considered a dead neuron, its connection to n_1 becomes a candidate for the "Forget" operation. Recognizing that W_1 holds the utmost importance in the network, we opt to hide it from potential attackers. Conversely, we select W_2 for the "Rewiring" operation and replace its value with that of W_1 as W_2 no longer affects the output of neuron n_1. To achieve this, we divide the activation from neuron m_1 by 2 (the division factor). Half of this division is passed to the same parameter W_1, and the other half replaces the activation of the dead neuron, going to W_2 (which now holds the value of W_1). This way, when attackers employ gradient search and loss backpropagation, they discover that the weight ranking has changed. With X_1 now reduced by half, it is smaller than X_3 making W_3 the most important parameter in the layer. It's worth noting that despite modifying a weight's value and rewiring activations, the output of neuron n_1 remains consistent, and the layer's functionality undergoes minimal change.

If an attacker arbitrarily targets a critical parameter (known to the developer, such as W_1), the attack is not as effective as in the non-FaR model. This is because W_2, to which the importance of W_1 has been redistributed, also needs to be targeted. Consequently, the model's resilience against random attacks is enhanced as well. We provide a comprehensive evaluation of the impact of our proposed strategy on Bit-Flip Attacks in the section 6.

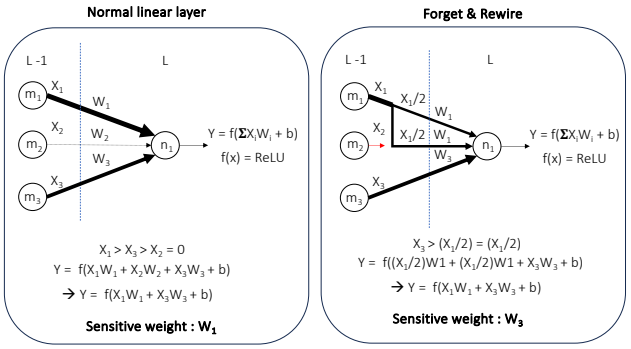

Figure 5: Forget and rewire example on a simple linear model.

4 Threat Model

In this work, our goal is tailored for defense against BFAs. BFA involves direct manipulation of the model's parameters at the deployment stage. This type of attack assumes that the attacker has the capability to alter the model directly, such as inducing bit-flips in the model's memory, which changes its behavior without needing to modify the input. Unlike BFAs, Adversarial Example Attacks primarily manipulate the input to the model without altering the model's parameters directly. The attackers' goal is to deceive the model into incorrect predictions while they may have complete knowledge of the model's architecture (white-box scenario). However, the model itself remains unaltered. Hence, the proposed FaR technique is specifically designed to mitigate the risks posed by BFAs, where the model's parameters are directly vulnerable to manipulation.

4.1 Assumptions

We assume the adversary possesses the capability to execute Bit-Flip Attacks against the models after their deployment. This involves the precise manipulation of multiple bits of multiple parameters to influence the prediction results of the model, not just single-bit-flips. This threat model represents a significantly stronger attack scenario. The feasibility and practicality of such threats have been previously validated and assessed in prior research [6, 25, 34, 37]. Additionally, the adversary has the potential to run their malicious program on the same machine as the victim model (on resource-sharing platforms such as ML-as-a-service) and employs methods such as DeepHammer to carry out BFAs.

Aligned with prior research [25, 43] focusing on DNNs, we adopt the white-box assumption, assuming that the adversary possesses the knowledge and capabilities necessary to execute a potent attack. More precisely, the adversary is presumed to have in-depth knowledge of the victim model, its parameters, and relevant details. Moreover, we extend this assumption to encompass the adversary's familiarity with every facet of any conceivable defense mechanism implemented within the system. This understanding involves the mechanism's operation, algorithms, parameters, and other relevant aspects. This knowledge can be acquired through methods like side-channel attacks [48].

We categorize attackers into three types based on their knowledge levels:

1. Basic Adversary: This attacker has no knowledge about the FaR defense strategy, and has no access to obtain the gradient values on the deployed model. However, the attacker has access to an equivalent model without FaR capability to evaluate its BFA on its local station.

2. Expert Adversary: This attacker is aware of FaR and is able to obtain the gradient values on the deployed model. However, the FaR configuration is not available to the attacker.

3. Oracle Adversary: This attacker is aware of FaR and not only is able to obtain the gradient values on the deployed model, the attacker has access to the FaR configuration.

While the Oracle attack is a very rare scenario, it is the strongest type of attack. It is important to note that the FaR configuration is not publicly available and will be stored securely, only to be applied during deployment. Obtaining such configurations would require an attacker to have an exceptionally high level of access, such as that of a malicious cloud provider [28]. Conversely, a model's architecture and parameters are often easily accessible to the public, as seen with models available on platforms like Hugging Face. Our objective is to enhance the model's robustness against more prevalent threats, such as RowPress and Deephammer, which attackers with varying levels of expertise (basic/expert) may exploit.

4.2 Defence Requirements

As mentioned before, in this work we seek to establish a defensive strategy to harden Transformers against a range of BFAs and potential adaptive threats. It's important to emphasize that our objective is not to completely eliminate BFAs but rather to increase the difficulty and cost associated with these attacks. Even with a robust defense in place, adversaries may still attempt to manipulate more parameter bits. Therefore, our aim is to significantly raise the number of bit-flips needed for adversaries to achieve their malicious goals, making the attack less practical and inconspicuous. The specific criteria for our defense strategy are as follows:

1. Our defense solution should have a minimal impact on the model's inference process. It must maintain the usability of the original model, ensuring that it retains its prediction accuracy to a significant extent without significant degradation.

2. Our defense approach should be easily applicable to standard, off-the-shelf Transformers. It should not require significant alterations to the original model's parameters, such as retraining the model from scratch, which can be computationally expensive.

3. Our goal is to implement a solution at the application level that is effective across various hardware architectures, operating systems, and deep learning libraries. In contrast to previous works that have proposed hardware or system-level measures to counteract fault injection attacks, our solution should not be limited to specific platforms.

5 Forget and Rewire

In this section, we introduce both the theoretical principles and practical implementation guidelines for our rewiring strategy. These principles ensure the effectiveness of our proposed approach in countering adversarial bit-flips and establish a robust foundation for the detailed defense implementation covered in the subsequent section.

5.1 Theoretical Basis

To thwart an attacker attempting to flip weight bits on important parameters, we analyze the attacker's strategy, which often involves a greedy approach to identify the most vulnerable weight bits in the model. While the attacker's choice of bit-flip sequence may be influenced by the input, the initial crucial step is to identify a bit with the highest gradient based on the loss function, on a batch of input data, for each layer. Subsequently, the attacker assesses the impact of individual bit-flips on the loss function and iteratively selects the most influential bit to flip in each iteration.

In essence, if we deliberately distribute the importance of critical weights among others, we can enhance the likelihood of preventing bit-flips to those crucial weights, rendering the attacker's bit-flip sequence ineffective. Therefore, we delve into the backpropagation rule for computing the gradient of the loss/cost function C with respect to weight W_{mn}. This weight connects neuron m at layer $l-1$ to neuron n at layer l. The training is the process of minimizing error/cost (which originates from the difference between the output of the model and the desired output). The model's output is determined by the activation produced by the last layer neuron, which is a function of the neuron's parameters (weights): $a = f(W)$. The relationship between activation and weight is linear, computed as a weighted sum of inputs plus a constant bias. In the context of gradient descent training, the goal is to minimize the error by moving along the cost function, in the direction of the slope of C (the derivative of the cost function). Since the formula for the cost function is typically available (with the simplest form being Mean Squared Error), we can readily calculate $\frac{\partial C}{\partial a}$. However, during the attack, the attacker is interested in the weights W. Therefore, we require an expression for the rate of change of the cost function concerning W, which necessitates the application of the chain rule of differentiation: $\frac{\partial}{\partial x} f(g(x)) = g'(x)f'(g(x))$

The attackers' objective is to determine the extent to which they can maximize the error by corrupting the weights W. To achieve this, we need to realize how changes in W impact the output activation, and subsequently, how alterations in the output activation influence the cost function. This can be elucidated as follows: $\frac{\partial C}{\partial W} = \frac{\partial a}{\partial W} \frac{\partial C}{\partial a}$

Building upon the preceding discussion, $\frac{\partial C}{\partial a}$ is readily accessible, and the derivative of activation w.r.t. W is simply the input to the parameter. This approach remains consistent for the hidden layers, as illustrated in Figure 6. $\frac{\partial C}{\partial W}$ likewise signifies how responsive (sensitive) the output is to the parameter W. Through this method, we can discern the degree of influence each parameter in the model has on the output.

Now, we can put it all together in the following equation:

$$\frac{\partial C}{\partial W^l_{mn}} = x_{mn} f'(a^l_n) \Sigma_{i=1}^{k^{l+1}} W^{l+1}_{ni} \frac{\partial C}{\partial a^{l+1}_i}$$

where x_{mn} is the input to the parameter connected to neuron n from neuron m at layer $l-1$. a^l_n is the neuron n's output before passing to its activation function f (such as ReLU). Σ term is the weighted summation of loss of all neurons at layer $l+1$.

The derivative $\frac{\partial C}{\partial W^l_{mn}}$ indicates how sensitive the output is to the weight parameter W. A higher value suggests that this parameter can induce more significant changes in the output (resulting in model loss), which makes it an attractive target for attackers. The goal is to diminish this sensitivity while preserving the layer's functionality. To achieve this, we need to identify which term plays a crucial role in determining the magnitude of this derivative. Typically, $f'(a^l_n)$ remains a constant value (0 or 1), especially with widely used ReLU functions. The summation term (Σ) doesn't necessarily decrease the absolute value because weight values and loss terms can be either negative or positive. However, the activation value from neuron m to neuron n can have the most significant impact. Simply put, by reducing x_{mn}, we can decrease the gradient magnitude of all weights originating from source neuron m to any destination neuron n. This principle also applies to the output layer.

In this section, we detailed our method for identifying critical parameters within a model, emphasizing that the activation from the preceding layer influences neuron activation more than the weight of the parameters themselves. This approach leads us to rank parameters based on their gradients, as illustrated in the simple example in Figure 6. The type of activation function, whether ReLU or GELU, does not affect this process. It is notable that our analysis does not imply linearity in neural networks; rather, it highlights how Transformer models' linear layers facilitate the FaR operation.

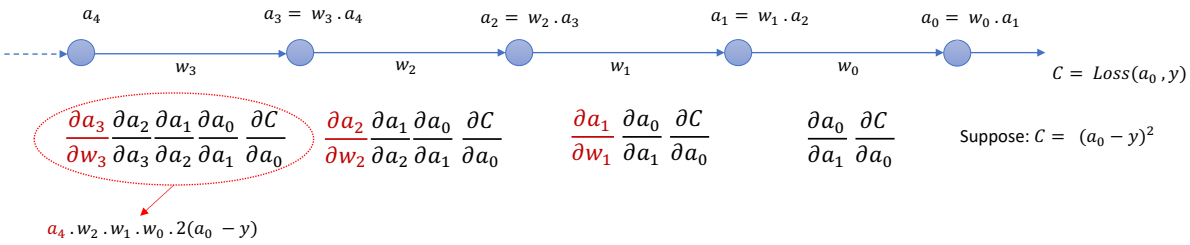

Figure 6: Calculating the sensitivity of output to weights.

5.2 Strategy Principles

Our proposed approach revolves around two key actions: *Forget* and *Rewire*. Additionally, we introduce a hyperparameter known as the *division factor*. The Forget operation involves disengaging an activation generated by a *dead* neuron, meaning it will not be transmitted to other neurons, and its weight parameter will be replaced with a new value. A dead neuron, in this context, is one whose activation hovers around zero or is very close to it when assessed using a batch of input data. Such neurons have minimal impact on the model's functionality. This concept aligns with the well-established "Pruning technique" which typically disregards around 10 to 15% of neurons in a linear layer without compromising model accuracy.

On the other hand, the Rewire action redirects an activation that contributes to a parameter's importance to another parameter. During this process, the magnitude of the activation is divided by the division factor. The division factor plays an important role in determining the complexity and effectiveness of our strategy. An increase in this factor may lead to the utilization of multiple neurons for redundancy, potentially diminishing the model's generalizability and risking a reduction in accuracy. Conversely, a smaller division factor may not adequately achieve the concealing objective, thus lacking sufficient robustness. We will assess the impact of this factor in detail in the Evaluation Section.

To implement our strategy, as shown in Figure 7, we first need to identify the crucial weight parameters within the linear layers of the model and devise a plan to conceal them from potential attackers. To achieve this, we utilize a batch of input data and perform loss backpropagation to compute gradients, subsequently sorting them to rank the parameters. Once an important parameter is identified in a layer, our next step is to locate one or more dead neurons (based on a division factor) within that same layer. When a dead neuron is pinpointed, we subject its activation to the Forget operation, effectively disconnecting it from the neural network. Simultaneously, we select its connection to the neuron containing the important parameter for the Rewiring operation. In this process, the activation directed toward the important parameter is divided by the division factor before being rerouted to the forgotten parameter. Consequently, the value of the forgotten parameter is overwritten by that of the important parameter. This process can be iterated until a sufficient number of parameters are obfuscated, consequently raising the number of bit-flips required to significantly degrade the model's accuracy. It's important to emphasize that after each forgetting and rewiring step, the entire gradient calculation process must be repeated.

5.3 Algorithm for Application

Algorithm 1 outlines a systematic procedure aimed at augmenting the resilience of Transformer-based models against Bit-Flip Attacks. Initially, the model iterates through a predefined number of steps, each representing a single FaR operation. In each iteration, the algorithm first identifies the most crucial parameter within the model by computing and analyzing gradients with respect to the input data. This parameter, deemed significant due to its gradient value, is marked for protection against potential attacks.

Subsequently, the algorithm seeks out dead neurons that contribute minimally to the model's output due to their negligible activation levels. These neurons are pinpointed by inversely sorting the computed gradients, thereby revealing the least impactful parameters in the model's architecture.

Once the critical and dead parameters are identified, the algorithm devises a Forget and Rewire (FaR) configuration. This configuration essentially outlines a strategy for each layer of the model, determining how the activations of dead neurons should be rerouted and their associated weights adjusted to shield the important parameters. Since this configuration is small and does not include all parameters, it can be stored securely on-chip and not be available to public users.

The final step involves applying this FaR configuration to the model during the deployment. This is achieved by locating the specific neurons and layers as per the configuration, and then overloading its customized linear layer with the FaR configuration and weigh replacement. During runtime, the custom linear layer itself adjusts the neural connections accordingly. Notably, the activation directed towards a crucial parameter is rerouted to a previously inactive (dead) neuron, and its magnitude is moderated by a predefined division factor. This step not only camouflages the important parameters but also ensures the model's functional integrity by redistributing activations.

The entire process is iterative, with each cycle further fortifying the model's defense by obfuscating additional parame-

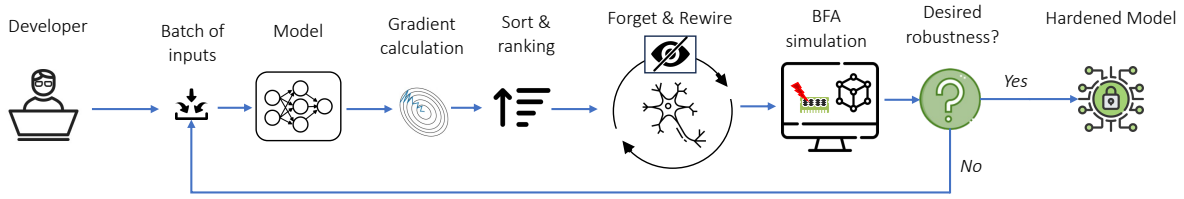

Figure 7: Forget and Rewire hardening framework.

Algorithm 1: Forget and Rewire Pseudo Code

Input: model, sample_data, labels, max_far
Output: Enhanced model

for i = *1 to max_far* **do**

Step 1: Identify Important Parameter

$important\_param\_d \leftarrow compute\_gradient\_wrt\_input(model, sample\_data, labels)$

Step 2: Locate Dead Neurons

$dead\_param\_d \leftarrow get\_dead\_params(model, sample\_data, labels, important\_param\_d)$

Step 3: Generate Forget and Rewire Configuration

$far\_cfg\_d \leftarrow get\_forget\_rewire\_cfg(important\_param\_d, dead\_param\_d, model)$

Step 4: Apply Forget and Rewire Configuration

$model \leftarrow apply\_far\_cfg\_to\_model(model, far\_cfg\_d)$

ters. Post each FaR operation, the model undergoes a reassessment of accuracy and BFA resistance. As soon as the accuracy is dropped or BFA resilience is achieved, the operation can be stopped.

In a network comprising millions of parameters, the division of activations assigned to weights has a substantial impact on the weights ranking within the model. Consequently, an attacker will only identify less important parameters as critical. As a result, they must perform a greater number of bit-flips to achieve the same level of accuracy reduction compared to a network lacking the forget and rewire defense strategy.

5.4 Security Analysis

To enhance the efficiency and efficacy of our strategy, we must explore several key aspects:

1. FaR Placement: It is required to investigate where to position the *Forget and Rewire* neurons, including determining the appropriate layer and specific neuron within a layer.

2. Activation Ranking Strategy: It is essential to develop a robust activation ranking strategy, which may involve ranking at different levels or stages within the model architecture.

3. Hyperparameter Tuning: Fine-tuning the hyperparameters is crucial, including selecting the optimal number of important parameters and determining the *division factor*.

Optimizing these elements will contribute to the overall performance of our strategy in mitigating Bit-Flip Attacks.

6 Evaluation

In this section, we provide overarching recommendations for the selection of pivotal hyperparameters within the Forget and Rewire defense strategy. Specifically, we explore the effects of these parameters, namely the choice of layer, the number of parameters, and the division factor, on our model's performance. We analyze the inference time, accuracy, size overhead, and BFA efforts.

6.1 Experimental setup

Our BFA analysis framework systematically flips individual bits within a model and quantifies the resulting impact using predefined metrics. The implementation of this framework is conducted in Python 3.11 and Torch 2.0.1, which supports CUDA 11.7 for GPU acceleration of computations. The experiments are conducted on an AMD EPYC 7302 CPU (16 physical cores), running at 3.0 GHz. This platform is equipped with four NVIDIA GeForce RTX 3070 GPUs.

We utilize the Yelp Open Dataset, which is available in JSON format for LLM analysis. Specifically, we focus on the *review.json* file, which contains comprehensive review text data.We also evaluate the effectiveness of our FaR-enabled defense framework with MNIST, CIFAR-10, CIFAR-100, and a subset of ImageNet (50 samples per class from validation set) for image classification [2].

For this study, we utilize the *transformers* package from *Hugging Face* [1] and its pre-trained models. We selected two machine learning models suitable for image and text classification tasks. Our selection is based on the most commonly used models ($google/vit-base-patch16-224$ on ImageNet with 86.56M parameters, and $dbmdz/bert-large-cased-finetuned-conll03-english$ on Yelp with 334M parameters). It should be noted that the *dbmdz* model is only

used for the token classification task. Three more custom ViT structures were also developed and trained on MNIST, CIFAR-10, and CIFAR-100 datasets. A comparison of custom models is presented in Table 1.

6.2 Model Integrity Evaluation

A paramount criterion for any effective hardening method is preserving the model's original functionality, ensuring minimal deviation from its trained objectives, and safeguarding against significant accuracy loss. The redistribution of critical weights to *dead neurons* is critical to maintaining accuracy. FaR ensures that essential information still traverses through both original and newly established detour pathways to the output layer, preserving critical data. Hence careful selection of dead neurons via gradient search to avoid data loss is an important step in FaR. Moreover, FaR is a one-shot process that does not necessitate retraining.

Table 2 presents a comparative analysis of the impact of FaR on various models and datasets. It's noteworthy that the amount of degradation is comparable with Aegis, incurring a reduction of 2% [43] on CIFAR-10/100. In contrast, alternative approaches like BIN [16] and RA-BN [38] exhibit more pronounced accuracy degradations, approximately ranging between 2 to 4%. It's also observed that the NeuroPots method demonstrates an accuracy comparable to our FaR approach, reinforcing the notion that the FaR mechanism maintains model integrity without substantially compromising its accuracy.

Our proposed mitigation strategy stands apart from retraining-based methodologies that necessitate complex and resource-intensive fine-tuning processes. Such fine-tuning is typically accessible to defenders with ample resources and access to large training datasets, especially for LLMs and ViTs. In contrast, our strategy offers a one-shot model modification approach that effectively addresses these challenges through straightforward computations, incurring minimal cost. Notably, it is observed that this approach has only a marginal impact on model accuracy, particularly when it involves rewiring critical parameters.

6.3 Model Size Evaluation

While the FaR configuration is securely stored separately from the model's parameters, applying its configuration can lead to an increment in the model's size during deployment. This increase has been systematically assessed across various datasets and model architectures utilized in our study, as shown in Table 3. Notably, this augmentation in size is predominantly influenced by the model's architecture and the quantity of parameters designated for the FaR operation, rather than by the datasets themselves. In particular, FaR's memory usage increases linearly with the number of FaR-enabled parameters.

It is important to underscore that such increments in size

Table 1: Custom ViTs for evaluation on selected Datasets

ViT Params	MNIST	CIFAR-10	CIFAR-100
image size	28	32	32
channel size	1	3	3
patch size	7	8	8
embed size	512	512	512
num heads	8	8	8
classes	10	10	100
num layers	1	3	6
hidden size	256	256	256
Model size	3.19M	9.57M	19.07M

Table 2: The accuracy (%) of models (w/o and w/ FaR) on various validation datasets and transformer structures

Dataset	w/o FaR	w FaR	Ageis	NeuroPots
MNIST	98.3	-0.1	—	—
CIFAR-10	96.1	-1.14	-1.26	-1.0
CIFAR-100	92.8	-1.35	-1.96	—
ImaegNet	88.4	-1.97	—	-1.3
Yelp review	Base	-1.82	—	—

generally are not a bottleneck for the practical deployment of models even on embedded devices. Common embedded devices, such as the Nvidia Jetson Nano, are typically equipped with memory capacities at the GB scale, amply sufficient to accommodate the FaR enhancement. Moreover, the FaR technique significantly bolsters robustness while maintaining accuracy levels, rendering the trade-off of a modest size increase highly beneficial, especially for edge-based inference applications.

To corroborate these assertions, we deployed our FaR-enhanced ViT model on a widely utilized edge device, the Nvidia Jetson Nano, equipped with 4GB of memory. The model's size increment proved manageable for this device since the average inference time increased only 4 to 6%. This underlines the practical viability and efficiency of the FaR approach on embedded devices.

6.4 Performance and Time Overhead

One of the key objectives of our design is to guarantee that the implementation of the FaR mechanism introduces minimal overhead to the operational framework of the model. Table 3 presents the timing metrics associated with model inference, both with and without the integration of FaR. It is evident that the time increment attributable to our approach remains marginal, primarily due to the incorporation of forget and rewire operations within the inference process. This additional time demonstrates a linear correlation with the volume of parameters earmarked for the FaR process.

The process of parameter selection for FaR is conducted offline, post-training. The duration of this process varies depending on several factors, including model size, computing resources (CPU/GPU), FaR configuration, and the desired

Table 3: Storage and Time overhead of FaR-based hardening on different Transformer architectures and datasets.

	MNIST ViT1	CIFAR-10 ViT2	CIFAR-100 ViT3	Yelp dbmdz	ImageNet google
Inference Time (%)	+0.0	+0.3	+0.1	+1.1	+2.5
Model Size (%)	+2.7	+3.3	+5.1	+8.9	+9.3
Offline FaR Calculation Time (H)	9	22	65	237	184

level of robustness against BFA. For instance, the time required to find FaR settings for a ViT trained on ImageNet can range from a few hours to several days as shown in Table 3.

Currently, the most significant contributor to the inference time overhead is the fact that our customized linear layer, infused with FaR configurations, lacks optimization with binary libraries and operates solely within a Python environment. This contrasts with the conventional linear layers in PyTorch, which benefit from optimized matrix multiplication operations. Addressing this, we aim to explore Python bindings in C++ for FaR, aspiring to expedite the FaR operations within the linear layer in future endeavors. While principles underlying FaR suggest its theoretical scalability to larger models, including LLMs with multi-billion parameter configurations, in practice, as the results show in Figure 13, an optimized FaR library is a must to achieve acceptable inference overhead for such enormous models.

6.5 Concealing Efficiency

The attack chain refers to the set of parameters targeted in a BFA, indicating which parameters were considered highly important from the attacker's perspective. *Concealing Efficiency* is the rate at which these parameters are concealed (rewired) within the attack chain.

Table 4 demonstrates the proficiency of our hardening technique in obfuscating pivotal parameters that are typically targeted in bit-flip chains orchestrated by BFA strategies. For instance, our fortified model successfully identifies and conceals 92%, 83%, and 77% of parameters in the basic, expert, and oracle attack chains, averaging a concealing rate of 84% in total. While it's acknowledged that our defense mechanism may not obscure every parameter from potential attackers, the FaR-enabled model retains its core functionality and resilience against attacks. This resilience is underpinned by our method's strategic focus on shielding the most critical parameters along the attack's critical path—parameters whose alteration could lead to significant compromise if included in a bit-flip attack chain.

Figure 8 further explores the influence of hyperparameters (elaborated upon in Section 6.9) on the concealing rate. The observations from increasing the number of layers in our experiments show a diminished effect on the concealing rate,

Table 4: Example of BFA chains generated on the model w and w/o FaR and conceal rate

Dataset	Attack	FaR	# of bit-flips	Coverage (%)	Accuracy drop (%)
MNIST	Basic	No	75	—	-32.3
		Yes	282	97	-31.4
	Expert	No	38	–	-31.7
		Yes	99	92	-32.2
CIFAR-10	Basic	No	112	—	-36.5
		Yes	371	89	-35.8
	Expert	No	86	—	-36.4
		Yes	210	82	-36.0
CIFAR-100	Basic	No	108	—	-27.1
		Yes	562	89	-27.7
	Expert	No	89	—	-28.3
		Yes	370	78	-28.4

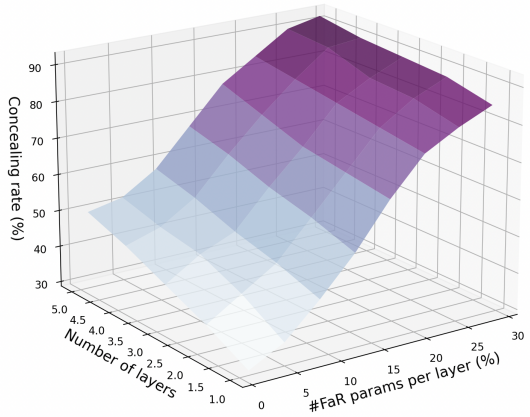

Figure 8: The impact of the number of FaR parameters per layer, and the number of selected layers on the concealing rate of important parameters in basic BFA's critical path.

contrasting with the effect observed when altering the number of FaR parameters per layer. This pattern implies that the majority of critical parameters, which affect the concealing rate, are located within the last few layers of the model.

6.6 Mitigation Effectiveness

Another notable advantage of our proposed strategy is its resilience even against an Oracle attacker with knowledge about the hardened model.

Figure 9 shows our analysis of the number of bit-flips required to degrade the accuracy of the FaR model by more than 10% when the decision factor is changed. The normalization is grounded on the bit-flips required for Oracle attacks on non-FaR models. A division factor of 1 indicates that each critical parameter is rewired with one dead neuron. Figure 9 depicts that even Oracle attackers require 1.68 times more bit-flips (division factor 5). Under the same configuration, the model's accuracy decreases by 1.83% when 10% of parameters per layer are selected for rewiring (Figure 10).

In contrast to NeuroPots, where an attacker aware of the honey neurons can bypass them by avoiding highly sensi-

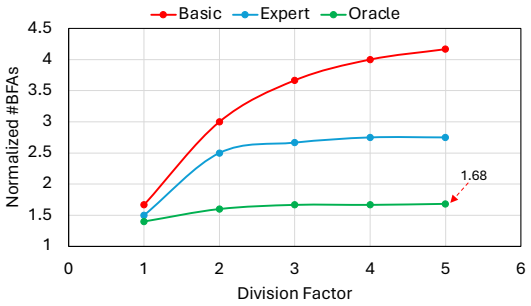

Figure 9: The impact of the division factor on the normalized number of bit-flips required to degrade the accuracy by 10% when 15% of parameters per layer are selected for FaR .

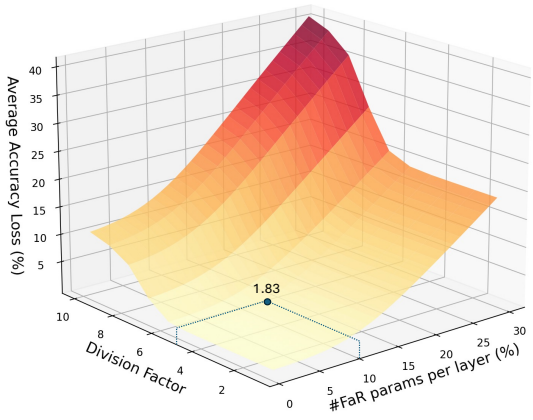

Figure 10: The impact of the number of FaR parameters and division factor per layer on the accuracy loss.

Table 5: Number of bit-flips (Mean and SD) required to degrade 10% of the accuracy of clean models and FaR models (10% per layer) under 20% Dropout (DO) during training and 20% Pruning (PR) post-training

FaR	Regularization	MNIST	CIFAR-10	CIFAR-100
w/o FaR	No DO - No PR	13 ±4	15 ±7	19 ±5
	DO - No PR	13 ±5	16 ±5	20 ±4
	DO - PR	11 ±2	14 ±4	17 ±3
w FaR	No DO - No PR	46 ±11	51 ±8	58 ±6
	DO - No PR	47 ±10	53 ±13	61 ±9
	DO - PR	39 ±6	48 ±9	56 ±8

tive parameters, our approach presents a more robust defense. Even if the attacker is informed about the obfuscation of critical parameters through the forget and rewiring schemes, they cannot exploit this knowledge effectively. This is because the critical parameter is no longer the single point of failure, and the model's critical path has been diversified, with tasks distributed among other neurons. This redistribution creates alternative pathways or *spare paths* via rewired neurons, allowing crucial information to continue propagating through the model even if the original vital parameters are compromised. Consequently, the attacker is still compelled to execute additional bit-flips to degrade the model's accuracy, making the attack considerably more challenging, resource-intensive, and exposed to detection.

Our hardening strategy can seamlessly integrate with various online fault detection and model recovery techniques. One straightforward detection mechanism involves checksum-based detection, applied selectively to a subset of parameters within each model. Following the implementation of the forget and rewire technique on the model, the parameter rankings shift compared to the original model. Less important parameters now assume higher ranks, rendering them more vulnerable to potential attacks by adversaries. Consequently, within each critical layer of the modified model, we can designate a subset of the highest-ranked parameters for checksum calculation, saving the results in a secure zone. During runtime, if the sum of the selected weights deviates from the original sum, it signals potential tampering with the model. Once faults are detected, we can promptly replace the weights of the affected layer with a clean copy in real-time. This integrated approach enhances the model's robustness against both attacks and faults.

6.7 Regularization and Pruning

Dropout [46] is a regularization technique used during the training phase, and randomly deactivates a subset of neurons to prevent the network from becoming overly dependent on specific neurons, thus enhancing generalization. However, during the inference phase, dropout is not active, and all neurons contribute to the output. Pruning [17] involves removing less significant neurons or connections either during or post-training. This process streamlines the network for inference, making it leaner and potentially faster but does not inherently protect against adversarial threats such as bit-flip attacks.

Table 5 presents a detailed comparison of pruning, dropout, and FaR on robustness against BFAs. Neither dropout nor pruning is designed to obscure or protect critical parameters from adversarial discovery and attacks. Pruning inadvertently makes critical parameters more discernible by simplifying the network, easing BFA by 11% on average. Dropout aims to prevent overfitting and does not offer specific protections against gradient-based adversarial attacks during inference. However, it could slightly enhance and increase the number of bit-flips by 4% on average.

6.8 Adversarial Examples

Although adversarial example attacks are not within the scope of our threat model [31], examining the impact of the FaR strategy on model robustness against such attacks offers valuable insights. To this end, we evaluated the resilience of FaR-hardened models against a well-known adversarial example attack, iterative FGSM (epsilon is set to 0.01) [20]. We analyzed

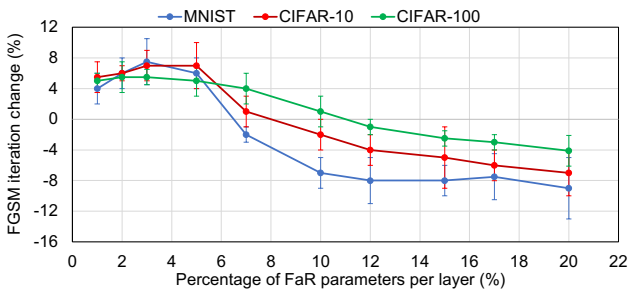

Figure 11: The impact of the number of FaR parameters per layer on the number of iterations required to generate an adversarial example to fool the model.

how reconfiguring model parameters through FaR influences the generation and effectiveness of adversarial examples on ViT models trained on MNIST, CIFAR-10, and CIFAR-100. We randomly selected 100 images from the validation dataset and applied the attack on FaR models. The results, presented in Figure 11, show that limited FaR operations can slightly increase robustness against adversarial perturbations. However, as the number of FaR operations per layer increases, we observed a reduction in the number of iterations required to fool the model. This finding indicates a delicate trade-off between resilience to BFA and susceptibility to adversarial examples. While a higher number of FaR operations enhances resilience against BFA, it can also increase vulnerability to adversarial examples, likely because rewiring parameters may exacerbate the propagation of perturbations within the model. Another observation is that by increasing the model size (from MNIST to CIFAR-100), the impact of FaR on adversarial examples' strength is reduced.

6.9 Ablation Study

In this section, we examine the effects of FaR's hyperparameters (particularly the number of parameters selected for FaR per layer and the division factor) on the performance of Transformer models, specifically focusing on our custom ViT trained on the CIFAR-100 dataset.

Figure 10 illustrates the effect of FaR hyperparameters on the accuracy degradation of our custom ViT. In this study, we varied the percentage of parameters considered for FaR operations per layer from 1% to 30% of the total parameters in a layer. We also incrementally adjusted the division factor from 1 to 10.

The findings reveal that an increase in both the number of parameters involved in FaR operations and the division factor leads to a reduction in accuracy. This outcome is attributed to the fact that rewiring a larger number of parameters compromises the model's generalizability. Essentially, the model becomes overly optimized for resisting BFAs specific to the classes of inputs used in gradient calculation, at the expense of its broader applicability.

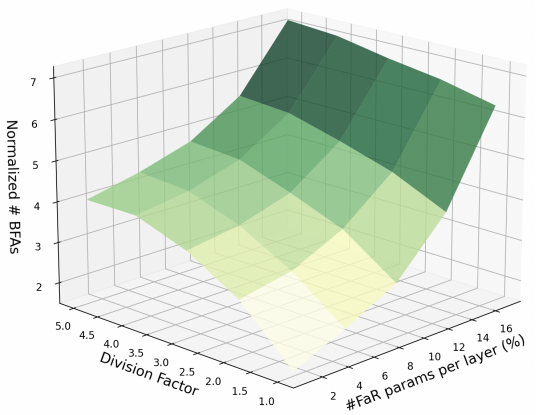

Figure 12: The impact of the number of FaR parameters and division factor per layer on the normalized number of bit-flips (using basic attack) required to degrade the accuracy by 10%.

The accuracy also proves to be highly dependent on the selection of input batches for gradient computation to identify critical parameters. Varying the classes of input samples used in this process would likely alter the parameters deemed most important, thereby affecting the model's validation set accuracy. Our experiments show that diversifying the input batches used to determine significant parameters mitigates the adverse effect on accuracy. However, this strategy may not necessarily enhance the model's overall resilience to generic BFAs but could reduce its defense against targeted BFAs aimed at specific output classes.

Initially, an increase in the division factor does not adversely affect accuracy due to the strategic selection of dead neurons for rewiring. Nonetheless, a notable decrease in accuracy is observed when expanding the scope of parameters classified as dead neurons, particularly when such expansion causes the FaR mechanism to contradict its intended function. This analysis shows that there is a delicate trade-off between FaR parameters and the accuracy of the model.

Figure 12 illustrates the role of FaR hyperparameters in strengthening models against basic BFAs. The findings reveal that an increase in the number of parameters involved in the FaR process leads to a quadratic rise in the number of bit-flips necessary to reduce the model's accuracy to the level of random guesses (representing a 10% loss, with the baseline being the count of BFAs in scenarios not employing FaR).

Furthermore, Figure 13 depicts the influence of the number of FaR parameters and division factor per layer on the inference time overhead. It reveals an anticipated trend: a rise in inference time correlating with an increase in the number of FaR parameters and division factors. It is notable that this additional time overhead is potentially reducible through the implementation of an improved and optimized linear layer. Such a layer would facilitate FaR operations via quicker C/C++ bindings within PyTorch, a modification we aim to explore in our forthcoming research efforts.

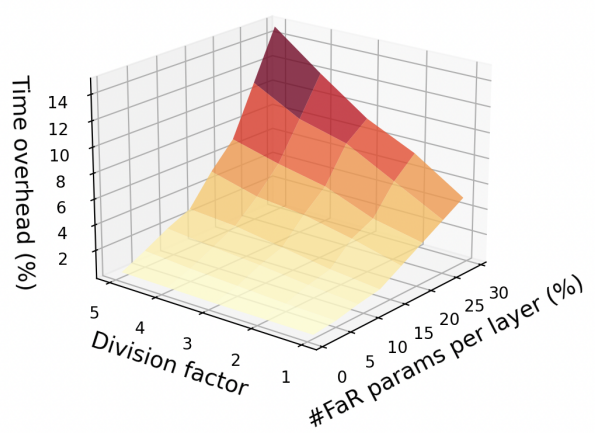

Figure 13: The impact of the number of FaR parameters and division factor per layer on the inference time overhead.

7 Discussion

7.1 Efficacy Against Sophisticated Attacks

Attacks Adapted against Gradient-based Defense: BFA aims to minimize the number of bit-flips required to degrade model accuracy, by targeting the most influential parameters that invariably exhibit high activation levels, making them detectable by gradient-based search. Consequently, our defense mechanism's effectiveness remains intact across different strategies for targeting critical parameters.

Attacks Adapted against Redistribution Strategy: Our defensive strategy, by design, increases the computational and operational cost for an attacker by necessitating a higher number of bit-flips. Should an attacker become aware of our rewiring scheme and attempt to target the original critical parameters, they would face the challenge of having to compromise all or most of the parameters that have been redistributed to a particular neuron. This is because attacking only the originally important parameter would not result in the expected performance degradation, as the activation is now also being propagated through other rewired parameters. This scenario forces the attacker to commit more bit-flips to effectively disrupt the model, aligning with our primary objective of enhancing resilience.

7.2 Using encryption to enhance security

Use of Authenticated Encryption: By applying authenticated encryption techniques to LLM memory, any tampering or bit-flips, such as those caused by row hammer attacks, become immediately apparent upon decryption. This ensures that any unauthorized alteration of the encrypted data can be detected, thus adding a robust layer of security.

Integration of ECC with Encryption: Extending the security measures, the incorporation of Error Correcting Code (ECC) alongside encryption can correct multiple-bit-flips. This is particularly beneficial in environments where data integrity and accuracy are critical. Although ECC can effectively manage multi-bit corrections, the scalability of this approach in the context of LLMs, which utilize billions of parameters, presents substantial challenges.

Resource Intensity and Feasibility: While the combined use of authenticated encryption and ECC offers a high level of security, the resource demands of implementing such comprehensive protection for billions of parameters are immense. Given the current technological and economic landscape, this solution might not be the preferred choice due to its significant resource consumption and associated costs that require further investigation.

7.3 Limitation and Future Research Directions

The current research has focused primarily on the application of our Forget and Rewire technique to linear layers within transformer-based models. However, there remain several promising avenues for future exploration and expansion of this work.

Potential Impact on Model Interpretability: The FaR technique, by design, involves redistributing critical weights across a broader spectrum of neurons, which inherently involves more parts of the network in performing similar tasks. This redistribution could potentially obscure the clarity with which specific network decisions can be attributed to particular neurons or layers. As a result, the effectiveness of techniques like Grad-CAM [40] and Saliency Maps [3], which rely on clear, localized activation patterns to interpret model decisions, might be diminished. While the potential impact on interpretability is valuable, it is important to note that the current scope of our evaluation does not include a comprehensive analysis of interpretability changes due to the implementation of FaR. Investigating the trade-offs between enhanced security and interpretability could provide valuable insights into the design and optimization of secure machine learning systems further studies could explore this dimension.

Extension to Convolutional and Recurrent Layers: One notable direction for future research involves adapting our Forget and Rewire strategy to convolutional layers (CNNs) and recurrent layers. While our current study centered on transformers, the applicability and potential benefits of our techniques in conventional deep neural networks (DNNs) like ResNet and AlexNet warrant investigation. Exploring how Forget and Rewire can enhance the robustness and security of these widely-used architectures is an important next step.

Integration with Other Defensive Techniques: Another intriguing avenue is the integration of Forget and Rewire with existing defensive techniques such as NeuroPots and Aegis. Combining these methods could lead to synergistic effects, bolstering the model's resilience against a broader spectrum of attacks. NeuroPots, serving as a detection mechanism, could

complement Forget and Rewire's protective capabilities, while Aegis, with its randomization approach, could add an extra layer of hardening to the model. Evaluating the effectiveness of these integrated defenses would be an insightful research direction.

Exploration of Non-Gradient-Based Attacks: While our work has primarily focused on gradient-based attacks, future research could explore the development of new fault injection attack methods that do not rely on gradient information. Investigating how Forget and Rewire stands up against such non-gradient-based attacks would provide a more comprehensive assessment of its security benefits.

Transferability and Prompt Tuning: Additionally, it would be valuable to analyze the transferability of our Forget and Rewire approach in the context of large language models (LLMs). Prompt tuning techniques have gained prominence in fine-tuning LLMs for specific tasks. Evaluating how Forget and Rewire can be adapted and integrated into prompt-based fine-tuning processes to enhance LLM security would be an intriguing research avenue.

8 Related Work

Machine learning models are susceptible to various types of attacks, including those focused on manipulating the data or the model itself [7,39,41,52]. Defenses against these attacks have been extensively studied for Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), including evasion, poisoning, backdoor/trojan attacks, and hardware-based fault injection techniques. However, the same level of attention and research for defense has not been devoted to transformer-based models. In the case of BFAs, which are a specific type of model-centric attack, studies proposing mitigation strategies in DNNs can be divided into two categories:

1. Hardening the Model: These approaches aim to make the model more robust and resistant to BFAs. They often involve architectural modifications or training techniques to fortify the model against potential attacks.

2. Detecting the Attack: These strategies focus on identifying when a BFA is occurring. They seek to create mechanisms or algorithms that can detect and alert when the model is under attack, allowing for a timely response to mitigate the damage.

In the first category of defenses, we can highlight the work of Li et al. [23], which employs a weight reconstruction method. This method helps neutralize the altered values in several model parameters, thereby reducing the impact of BFAs. Another notable contribution is from Zhan et al. [51], who modified the rectified linear unit (ReLU), a commonly used activation function in DNNs, to make it more resilient to faults caused by bit-flipping in weights. Additionally, there are binarization strategies such as BIN [16] and RA-BNN [38], which have been shown to be more effective than the aforementioned methods. These strategies involve retraining a binarization model from scratch to mitigate the effects of BFAs. However, it's important to note that these methods still require retraining the model, incurring substantial computational costs. Moreover, aggressive quantization, as seen in these approaches, can impact the model's accuracy.

The most recent work falling within the model hardening category is Aegis [43]. Aegis addresses targeted attacks, which aim to flip crucial bits in specific model layers. To thwart this, the authors introduce a dynamic exit mechanism with additional internal classifiers (ICs) in hidden layers. This disrupts attackers by allowing input samples to exit layers prematurely. The dynamic exit mechanism randomly selects ICs for predictions during each inference, making adaptive attacks more costly. They also propose robustness training for ICs, increasing model resilience by BFAs during IC training.

The second category operates independently of model improvement and offers an additional layer of protection. This category for defending against BFAs involves integrity verification [14,19,22,24]. In this approach, the defender generates a reference signature from the model before it is deployed. After deployment, new signatures are generated during inference and compared to the reference signature to detect any discrepancies.

The latest work that proposed a checksum-based BFA detection is NeuroPots [25]. NeuroPots introduces *honey neurons* as intentional vulnerabilities in DNN models, strategically placed to lure attackers into injecting faults by increasing the sensitivity of the model to less important parameters. This forms the basis for a defense framework with trapdoor capabilities. To efficiently detect faults, a checksum-based approach is used, focusing on the trapdoors where most bit-flips occur. Model accuracy is restored by *refreshing* these faulty trapdoors. A potential weakness of this approach arises when the attacker is aware of its implementation. In such cases, the attacker could circumvent trapdoors by disregarding sensitive parameters with sensitivity exceeding a predefined threshold.

The challenge at hand was to create robust defenses and detection methods uniquely suited to transformer-based models, which have distinctive architecture and traits. Our work distinguishes itself from previous research in three key aspects: 1) We introduce the concept of rewiring in a novel manner that eliminates the need for retraining. 2) Instead of exposing less crucial parameters as bait for attackers, we not only conceal important parameters but also enhance the model's resilience against BFAs by rewiring them to other parameters. 3) Our approach complements existing methods and can be employed alongside them, offering an additional layer of protection.

9 Conclusion

In addressing the imperative challenge of Bit-Flip Attacks (BFAs), this study illuminates a critical gap in research about the impact of BFAs on Large Language Models (LLMs) and Vision Transformers (ViTs). Inspired by the concept of "brain rewiring", we have proposed an innovative defense mechanism, *Forget and Rewire (FaR)*, specialized to enhance the resilience of Transformers against bit-flip attacks. FaR entails redistributing tasks from critical to non-essential neurons, effectively diminishing the model's sensitivity to specific parameters while preserving its core functionality. Despite BFAs' complexity, our proposed defense effectively shields sensitive weights, with minimal time and storage overhead while maintaining the model accuracy. The experimental results indicate the efficacy of our proposed approach, greatly decreasing the success rate of BFA attacks and obtaining a reduction factor ranging between 1.4 and 4.2 times, with minimal accuracy loss below 2%. Our novel FaR defense stands as a promising direction in confronting evolving digital threats, potentially steering the future of deep learning security solutions.

Acknowledgments

The work in this paper is partially supported by the National Science Foundation grants CNS-2155002, CNS-2155029, and CCF-2212427.

References

[1] *https://huggingface.com*.

[2] *https://pytorch.org/vision/stable/datasets.html*.

[3] Ahmed Alqaraawi, Martin Schuessler, Philipp Weiß, Enrico Costanza, and Nadia Berthouze. Evaluating saliency map explanations for convolutional neural networks: a user study. In *Proceedings of the 25th international conference on intelligent user interfaces*, pages 275–285, 2020.

[4] Asmita Asmita, Yaroslav Oliinyk, Michael Scott, Ryan Tsang, Chongzhou Fang, Houman Homayoun, et al. Fuzzing busybox: Leveraging llm and crash reuse for embedded bug unearthing. In *33th USENIX Security Symposium*, 2024.

[5] Yutong Bai, Jieru Mei, Alan L Yuille, and Cihang Xie. Are transformers more robust than cnns? *Advances in neural information processing systems*, 34:26831–26843, 2021.

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

[7] Kunbei Cai, Md Hafizul Islam Chowdhuryy, Zhenkai Zhang, and Fan Yao. Seeds of seed: Nmt-stroke: Diverting neural machine translation through hardware-based faults. In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, pages 76–82. IEEE, 2021.

[8] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[9] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 2023.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[11] Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, et al. Large language models for code analysis: Do llms really do their job? In *33th USENIX Security Symposium*, 2024.

[12] Tommaso Frassetto, Patrick Jauernig, Christopher Liebchen, and Ahmad-Reza Sadeghi. Imix:in-process memory isolation extension. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 83–97, 2018.

[13] Rodney A Gabriel, Brian H Park, Soraya Mehdipour, Dale N Bongbong, Sierra Simpson, and Ruth S Waterman. Leveraging a natural language processing model (transformers) on electronic medical record notes to classify persistent opioid use after surgery. *Anesthesia & Analgesia*, pages 10–1213, 2023.

[14] Yanan Guo, Liang Liu, Yueqiang Cheng, Youtao Zhang, and Jun Yang. Modelshield: A generic and portable framework extension for defending bit-flip based adversarial weight attacks. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 559–562. IEEE, 2021.

[15] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer.

IEEE transactions on pattern analysis and machine intelligence, 45(1):87–110, 2022.

[16] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. Defending and harnessing the bit-flip based adversarial weight attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14095–14103, 2020.

[17] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

[18] 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.

[19] Mojan Javaheripi and Farinaz Koushanfar. Hashtag: Hash signatures for online detection of fault-injection attacks on deep neural networks. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[20] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[21] Khang Nhut Lam, Loc Huu Nguy, Jugal Kalita, et al. A transformer-based educational virtual assistant using diacriticized latin script. *IEEE Access*, 2023.

[22] Jingtao Li, Adnan Siraj Rakin, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. Radar: Run-time adversarial weight attack detection and accuracy recovery. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 790–795. IEEE, 2021.

[23] Jingtao Li, Adnan Siraj Rakin, Yan Xiong, Liangliang Chang, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. Defending bit-flip attack through dnn weight reconstruction. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[24] Qi Liu, Wujie Wen, and Yanzhi Wang. Concurrent weight encoding-based detection for bit-flip attack on neural network accelerators. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2020*, 2020.

[25] Qi Liu, Jieming Yin, Wujie Wen, Chengmo Yang, and Shi Sha. Neuropots: Realtime proactive defense against bit-flip attacks in neural networks. In *32th USENIX Security Symposium (USENIX Security 23)*, pages 1–18, 2023.

[26] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. Rowpress: Amplifying read disturbance in modern dram chips. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–18, 2023.

[27] Kaleel Mahmood, Rigel Mahmood, and Marten Van Dijk. On the robustness of vision transformers to adversarial examples. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7838–7847, 2021.

[28] Hosein Mohammadi Makrani, Hossein Sayadi, Najmeh Nazari, Khaled N Khasawneh, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Cloak & co-locate: Adversarial railroading of resource sharing-based attacks on the cloud. In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, pages 1–13. IEEE, 2021.

[29] Anna McLean. Neurons that fire together, wire together. *Coach & Mentor*, page 2, 2020.

[30] Onur Mutlu and Jeremie S Kim. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8):1555–1571, 2019.

[31] Najmeh Nazari, Hosein Mohammadi Makrani, Chongzhou Fang, Behnam Omidi, Setareh Rafatirad, Hossein Sayadi, Khaled N Khasawneh, and Houman Homayoun. Adversarial attacks against machine learning-based resource provisioning systems. *IEEE Micro*, 2023.

[32] Cheng Qian, Ming Zhang, Yuanping Nie, Shuaibing Lu, and Huayang Cao. A survey of bit-flip attacks on deep neural network and corresponding defense methods. *Electronics*, 12(4):853, 2023.

[33] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. OpenAI, 2018.

[34] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1157–1174. IEEE, 2022.

[35] 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.

[36] 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.

[37] Adnan Siraj Rakin, Yukui Luo, Xiaolin Xu, and Deliang Fan. {Deep-Dup}: An adversarial weight duplication attack framework to crush deep neural network in {Multi-Tenant}{FPGA}. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1919–1936, 2021.

[38] Adnan Siraj Rakin, Li Yang, Jingtao Li, Fan Yao, Chaitali Chakrabarti, Yu Cao, Jae-sun Seo, and Deliang Fan. Ra-bnn: Constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy. *arXiv preprint arXiv:2103.13813*, 2021.

[39] Bita Darvish Rouani, Mohammad Samragh, Tara Javidi, and Farinaz Koushanfar. Safe machine learning and defeating adversarial attacks. *IEEE Security & Privacy*, 17(2):31–38, 2019.

[40] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[41] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In *30th USENIX security symposium (USENIX security 21)*, pages 1487–1504, 2021.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[43] 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 1–18, 2023.

[44] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. Learning deep transformer models for machine translation. *arXiv preprint arXiv:1906.01787*, 2019.

[45] Zhipeng Wei, Jingjing Chen, Micah Goldblum, Zuxuan Wu, Tom Goldstein, and Yu-Gang Jiang. Towards transferable adversarial attacks on vision transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2668–2676, 2022.

[46] Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, Tie-Yan Liu, et al. R-drop: Regularized dropout for neural networks. *Advances in Neural Information Processing Systems*, 34:10890–10905, 2021.

[47] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In *international conference on machine learning*, pages 1689–1698, 2015.

[48] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2003–2020, 2020.

[49] 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 1–18, 2020.

[50] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, et al. Scaling autoregressive models for content-rich text-to-image generation. *arXiv preprint arXiv:2206.10789*, 2(3):5, 2022.

[51] Jinyu Zhan, Ruoxu Sun, Wei Jiang, Yucheng Jiang, Xunzhao Yin, and Cheng Zhuo. Improving fault tolerance for reliable dnn using boundary-aware activation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(10):3414–3425, 2021.

[52] Mengxin Zheng, Qian Lou, and Lei Jiang. Trojvit: Trojan insertion in vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4025–4034, 2023.

[53] Kaijie Zhu, Jindong Wang, Jiaheng Zhou, Zichen Wang, Hao Chen, Yidong Wang, Linyi Yang, Wei Ye, Neil Zhenqiang Gong, Yue Zhang, et al. Promptbench: Towards evaluating the robustness of large language models on adversarial prompts. *arXiv preprint arXiv:2306.04528*, 2023.

[54] Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.