Bit-Flip Error Resilience in LLMs: A Comprehensive Analysis and Defense Framework

Yuhang Chen[1] Zhen Tan[2] Ajay Jaiswal[3] Huaizhi Qu[1] Xinyu Zhao[1] Qi Lin[2]
Yu Cheng[4] Andrew Kwong[1] Zhichao Cao[2] Tianlong Chen[1]
[1]The University of North Carolina at Chapel Hill [2]Arizona State University
[3]The University of Texas at Austin [4]The Chinese University of Hong Kong
{yuhang, huaizhiq, xinyu, andrew, tianlong}@cs.unc.edu
{ztan36, qlin36, Zhichao.Cao}@asu.edu
ajayjaiswal@utexas.edu chengyu05@gmail.com

Abstract

Bit-flip errors (BFEs) are hardware faults where individual bits in memory or processing units are unintentionally flipped. These errors pose a significant threat to neural network reliability because even small changes in model parameters can lead to large output shifts. Large language models (LLMs) are particularly vulnerable to resource-constrained or outdated hardware. Such hardware often lacks error-correction mechanisms and faces aging issues, leading to instability under the vast parameter counts and heavy computational loads of LLMs. While the impact of BFEs on traditional networks like CNNs is relatively well-studied, their effect on the complex architecture of transformers remains largely unexplored. *Firstly*, this paper presents a comprehensive systematic analysis of BFE vulnerabilities in key LLM components, revealing distinct sensitivities across parameters, activations, and gradients during fine-tuning and inference. *Secondly*, based on our findings, we introduce a novel defense strategy *FlipGuard*: (***i***) exponent bit protection, and (***ii***) a self-correction based fine-tuning mechanism, to address BFE consequences. *FlipGuard* minimizes performance degradation while significantly enhancing robustness against BFEs. Experiments demonstrate an average 9.27% reduction in accuracy drop under 1% BFEs on the SST-2 dataset using BERT, and an average 36.35-point improvement in perplexity on the Wikitext-103 dataset using GPT-2, compared to unprotected models. These results show the potential of our approach in enabling reliable LLM deployment on diverse and less reliable hardware platforms.

1 Introduction

Bit-flip Errors (BFEs) are hardware faults where individual bits in memory or processing units (e.g., GPUs) are unintentionally flipped from 0 to 1 or vice versa, as shown in Figure 1. While often linked to **aging** or resource-constrained hardware, recent

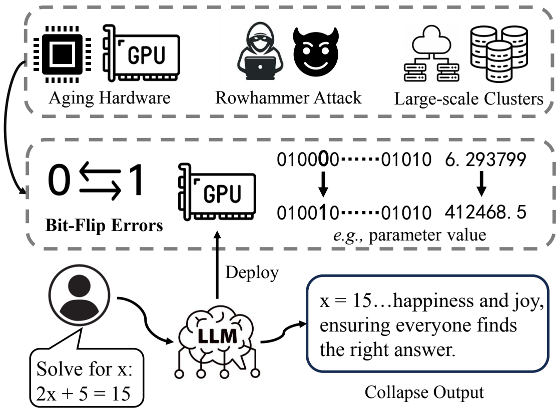


Figure 1: Illustration of how BFEs occur and impact LLMs. BFEs change values in GPUs, altering binary representations (*e.g.*, flipping 1 bit, value shifts from 6.293799 to 412468.5), which causes collapse outputs.

studies highlight that even minimal bit manipulations can severely degrade LLM performance or create backdoors (Liu et al., 2024), and BFEs in common LLM data formats like bfloat16 can cause up to 98% performance loss (Lhoussaine et al., 2024).

LLMs, with their vast parameter scales (Radford, 2018; Devlin, 2018) and high computational demands (Ajay Jain, 2024), exacerbate this vulnerability. Their deployment on diverse hardware, including older GPUs lacking robust error correction or specialized units prioritizing speed over fault tolerance (e.g., for bfloat16 operations), increases BFE risk. This leads to **instability** from frequent memory operations, making BFE investigation crucial for reliable LLM deployment. Besides, BFEs can also be intentionally induced by **attacks** such as Rowhammer (Kim et al., 2014), which exploits the increasing transistor density in DRAM, even in modern chips like DDR4 and DDR5 memory technologies (Frigo et al., 2020; Jattke et al., 2024). Moreover, the increasing reliance on *cloud infrastructure* for LLMs, particularly in large-scale clusters, amplifies this threat. The multi-tenant setup in public clouds, such as OpenAI's use of Microsoft Azure, shares memory hardware among clients, raising the risk of Rowhammer attacks and BFEs due to extensive data processing and computations (Xiao et al., 2016). *Therefore*, investigating BFEs in LLMs is essential for ensuring their reliability and robustness in real-world environments where hardware limitations are common.

Previous research has extensively explored the impact of BFEs on traditional neural networks, such as Convolutional Neural Networks (CNNs) (Breier et al., 2018). In these networks, BFEs can lead to significant drops in accuracy. However, these networks typically have fewer parameters and simpler architectures. In contrast, LLMs, with their massive parameter sizes and multi-layer structure, are prone to the propagation of errors. Despite this, the impact of BFEs on transformer-based LLMs remains under-explored. Given the increased vulnerability of LLMs due to their scale and complexity, we propose investigating how different components respond to BFEs in both fine-tuning and inference stages. This leads to the **first** critical research question:

> ***RQ1***: *How do bit-flip errors affect LLMs, given their unique architecture and scale?*

While some defense mechanisms have been proposed for smaller models, they are optimized for simpler architectures and are not directly applicable to LLMs. The massive scale of LLMs, coupled with intricate components like self-attention layers, requires novel strategies for detecting and mitigating BFEs. In addition, the diverse deployment environments for LLMs, including outdated hardware and potentially hostile cloud infrastructures, compounds the need for flexible solutions. This brings us to the **second** research question:

> ***RQ2:*** *How can we design effective strategies to mitigate BFEs in LLMs?*

To address ***RQ1***, we conduct a systematic analysis of the impact of BFEs on key components of LLMs during both fine-tuning and inference (Section 3). Specifically, we examine how BFEs affect critical modules, including ❶ self-attention layers, ❷ multi-layer perceptrons (MLPs), and ❸ embedding layers. Our analysis also explores the influence of bit-flips at different numerical positions, with a particular focus on the exponent's highest bit. We experimentally illustrate that errors induced in the most bit result in **disproportionately large** performance degradation due to its exponential effect on numerical values. Additionally, our study shows that different LLM components are subjected to **varying degrees of susceptibility** to BFEs. For example, embedding layers and layers closer to the input are more vulnerable because errors in these parts propagate throughout the network.

Based on the findings, for ***RQ2***, we propose a defense strategy: (***i***) enhanced quantization techniques to protect critical bits, and (***ii***) a self-correction based fine-tuning mechanism that exposes the model to random BFEs during fine-tuning, allowing it to learn corrective patterns. These strategies are designed to improve the robustness of LLMs against BFEs, improving reliable deployment across heterogeneous hardware environments, including outdated devices and cloud-based platforms susceptible to malicious attacks. We summarize our key contributions as follows:

- ***Error Investigation.*** We systematically analyze the impact of BFEs on key components of LLMs during fine-tuning and inference, providing a detailed understanding of how bit-flips at various stages affect LLMs performance.
- ***Defense Design.*** We propose a novel two-pronged defense strategy (*i.e.*, bit protection and self-correction mechanisms) to mitigate the effects of BFEs on LLM capabilities.
- ***Experiment Validation.*** We validate *FlipGuard* through extensive experiments, demonstrating significant improvements in robustness against BFEs. For example, our techniques can achieve a 36.35-point improvement in perplexity on the Wikitext-103 dataset using GPT-2.

2 Related Works

Bit-flip Errors and Defense in Neural Networks. Bit-flip attacks (BFAs) are closely related to bit-flip errors (BFEs), as both involve changes in bit values that can impact neural network performance (He et al., 2020). BFAs focus on *deliberately* inducing bit flips through methods like Rowhammer to exploit hardware vulnerabilities (Kim et al., 2014). Attackers employ techniques such as Progressive Bit Search to identify and flip critical bits, effectively degrading model performance with minimal perturbations. In contrast, BFEs occur randomly and unintentionally due to hardware faults, often caused by factors such as frequent access or aging memory components (Liu et al., 2023). While BFAs are an active area of research, the effects of BFEs in LLMs remain underexplored but pose a serious threat to model reliability on resource-constrained hardware.

Defenses against BFAs are categorized into two main approaches: fault tolerance and fault detection. Fault tolerance methods, including binarization-aware training, improve the model's resilience to BFEs but often come at the cost of reduced accuracy or increased computational overhead (He et al., 2020). RREC (Liu et al., 2022) uses redundant error-correcting codes to prevent BFEs propagation. NeuroPots (Liu et al., 2023) introduce honey neurons to attract and trap bit-flips. On the other hand, fault detection mechanisms monitor the system for bit-flips during runtime. Hardware-level defenses, such as SEC-DED (Hamming, 1950), can mitigate some attacks but remain susceptible to more advanced Rowhammer variants (Kwong et al., 2020). WRecon (Li et al., 2020) focused on recovering corrupted weights during inference by reconstructing weights affected by bit-flip errors.

Bit-flip Errors in LLMs. LLMs are vulnerable to numerical corruptions (Jiao et al., 2024; Mukherjee et al., 2003). Although there has been extensive research on BFEs in CNNs (Liu et al., 2023; He et al., 2020), the impact of bit-flip errors on LLMs remains unexplored. Meta has highlighted the importance of researching BFEs in LLMs (Jiao et al., 2024). Given the increasing scale of LLMs and their deployment in diverse hardware environments, it is crucial to investigate their vulnerability to BFEs, which is essential for ensuring the reliable deployment of LLMs in real-world applications.

3 LLM Serving with Bit-flip Errors

3.1 Bit-flip Error Setups

In this section, we define the BFEs simulation setups for LLMs. These errors arise naturally due to aging or resource-constrained hardware and occur unpredictably. The random nature of BFEs can differently impact key components, leading to different consequences. We aim to simulate the effects of BFEs through the components of LLMs.

Error Properties. We assume that BFEs randomly occur during fine-tuning or inference, affecting any bit of the stored or processed numerical data.

Error Scenarios. We define the following scenarios to simulate the occurrence of BFEs in LLMs:

- Inference-Time BFEs: These BFEs may lead to temporary incorrect outputs for a given prompt. A small amount of BFEs can potentially bring a dramatically different response.
- Fine-tuning BFEs: These BFEs may lead to permanent degradation in the model's learned parameters, degrading future inference.

Multi-Level Error Simulations. We conduct targeted simulations to evaluate the impact of BFEs at multiple levels of the model:

- Model Parameters: We simulate BFEs across layers, focusing on embedding, attention, MLP, and LayerNorm layers to reveal whether BFEs affect different components differently.
- Activations: We analyze BFEs in activations at different stages of the model, exploring whether BFEs have different impacts on stages closer to the input versus those closer to the output.
- Gradients: We examine gradient-level BFEs' impact during fine-tuning and the differences in BFE behavior between fine-tuning and inference.
- Bits Position: We simulate BFEs at different bit positions to identify which bit positions are more sensitive and critical to model stability.

3.2 Multi-Level Bit-Flip Error Simulation

3.2.1 BFE Impact on Model Parameters

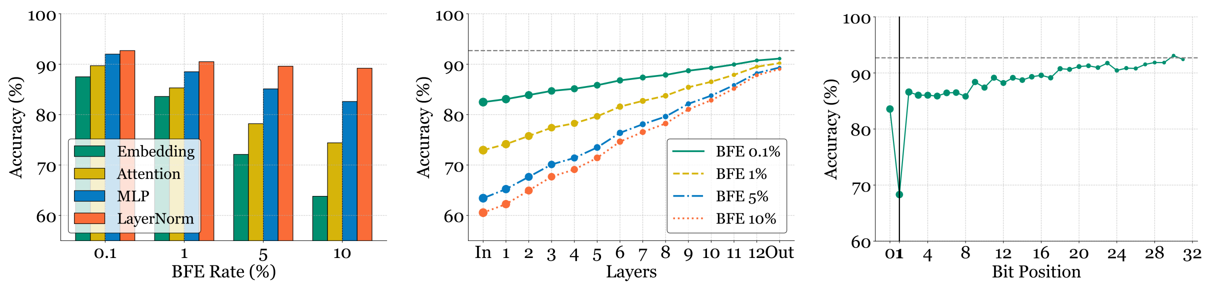
Observation 1 *BFEs in the parameter of the embedding layer have the most significant impact on the model's performance.*

In transformer-based LLMs, bit-flip errors can impact various components, including the embedding, self-attention, MLP, and LayerNorm layers. BFEs can disrupt token representations and alter attention scores, leading to performance degradation. This analysis helps us understand how errors in each layer differently affect model behavior.

Given a model with parameters $W = \{W^1, W^2, \dots, W^L\}$, where W^l are the parameters of the l-th layer. Each element W_i^l from layer l is a floating-point number with N_f-bit binary representation. The bit-flip error is introduced by flipping a randomly selected bit in the floating-point representation of W_i^l. For a randomly chosen bit position $k \in \{0, 1, \dots, N_f - 1\}$, the bit-flip error is injected using the following XOR operation:

$$\hat{W}_i^l = W_i^l \oplus 2^k \qquad (1)$$

where $\hat{W}_i^l$ is the perturbed weight after BFE, and $\oplus$ denotes the XOR operation. This formula applies BFE to the k-th bit of the binary representation.



(a) BFEs on parameters. (b) BFEs on activations. (c) BFEs on different bit position.

Figure 2: Performance degradation of GPT-2 on the SST-2 dataset under varying bit-flip error (BFE) rates. (a) illustrates the effect of BFEs on different model parameters (Embedding, Attention, MLP, and LayerNorm). (b) displays the accuracy changes when BFEs are applied to activations across different model layers. (c) shows the accuracy impact when BFEs are introduced at different bit positions within the model's parameters.

Thus, the error-injected model parameter set is:

$$\hat{W} = \{\hat{W}^1, \hat{W}^2, \dots, \hat{W}^L\} \tag{2}$$

We experiment with GPT-2 on the SST-2 dataset to measure the impact of BFEs across four different layers. BFEs are simulated at rates of {0.1%, 1%, 5%, 10%} in each type of layer. Figure 2(a) shows that BFEs in the embedding layer most significantly degrade performance. We argue this is because errors in embeddings propagate through the entire model. The attention and MLP layers exhibit moderate sensitivity, while LayerNorm is the least affected. These results highlight the importance of preserving embedding layer integrity to maintain model accuracy.

3.2.2 BFE Impact on Activations

Observation 2 *BFEs in activations near the input layer have more significant impact on model performance due to their larger cascading effect.*

In our analysis of how BFEs impact activations, we investigate errors introduced at different layers of the model: near the input, middle, and output. Activations represent the immediate output of a layer's computation, defined as: $A^l = f(W^l A^{l-1} + b^l)$ where A^l is the activation at layer l, W^l are the weights, b^l are the biases, and $f(\cdot)$ is the activation function. A BFE in the activation matrix A^l at position i is modeled as:

$$\hat{A}_i^l = A_i^l \oplus 2^k \tag{3}$$

where $\oplus$ is the XOR operation, and k is the bit position affected in the binary representation of the activation value A_i^l. This bit-flip error modifies the output of the layer and propagates forward:

$$A^{l+1} = f(W^{l+1}\hat{A}^l + b^{l+1}) \tag{4}$$

In Figure 2(b), we evaluate the impact of BFEs on activations using GPT-2 and the SST-2 dataset. We inject BFEs at various activation stages of the model. Errors in activations closer to the input layer have a cascading effect through subsequent layers, leading to widespread performance degradation. Conversely, errors in later activations have fewer impacts. This underscores the importance of detecting and mitigating errors in earlier activations to maintain model integrity.

3.2.3 BFE Impact on Gradients

Observation 3 *BFEs occurring in the fine-tuning stage will cause more long-term damage to model performance compared to inference-stage BFEs.*

BFE rate	FT-Grad.	FT-All	Infer.-All
0.1%	77.34	75.12	81.31
1%	71.88	69.89	75.61

Table 1: Accuracy results for different BFE rates on GPT-2 with SST-2 dataset under various experimental conditions. *FT-Grad.* refers to BFEs in gradients during fine-tuning, *FT-All* to BFEs in all values during fine-tuning, and *Infer.-All* to BFEs during inference.

Bit-flip errors during fine-tuning can have a far more detrimental effect than those occurring during inference. This is because BFEs in the fine-tuning stage can corrupt gradient updates, directly impacting model weights and leading to permanent performance degradation. These corrupted updates can either amplify parameter changes or render them ineffective, causing unstable training, poor convergence, and a model that is more vulnerable to errors in future inference. As a result, fine-tuning BFEs poses a higher long-term risk to model robustness.

The results in Table 1 confirm this trend and illustrate that BFEs during fine-tuning, particularly when they corrupt gradients, have a more pronounced and lasting impact on model performance than BFEs during inference.

3.2.4 BFE Impact on Bit Positions

Observation 4 *BFEs in the highest exponent bit lead to the most significant performance degradation due to the large magnitude shift they cause.*

Floating-point numbers in computing are typically represented using the IEEE 754 standard. For an N-bit floating-point number W_i^k, the bits are divided into three components: Sign Bit (S): b_1, Exponent Bits (E): $b_2, b_3, \ldots, b_{1+e}$, Mantissa (Fraction) Bits (M): $b_{2+e}, \ldots, b_N$. The floating-point value is calculated as:

$$W_i^l = (-1)^S \cdot 2^{E-E_{\text{bias}}} \cdot (1+M) \tag{5}$$

Where $E_{\text{bias}} = 2^{e-1} - 1$. For 32-bit single-precision floating-point numbers, S is b_1, E spans b_2 to b_9, and M spans b_{10} to b_{32}. In our experiments shown as Table 2, 99.9988% of GPT-2 parameters lie within $[-(2-2^{-23}), 2-2^{-23}]$, meaning the highest exponent bit, b_2, is typically 0. When $b_2 = 0$, the exponent $E - E_{\text{bias}}$ can range from -127 to 0, constraining $W_{i,j}^l$ within $[-2, 2]$. Since the maximum mantissa value $M_{\max}$ is less than 1, even when other bits change, parameters remain within $[-2, 2]$ range. If b_2 is flipped from 0 to 1, the exponent increases by 2^7, causing $E' = E + 128$, which leads to extreme parameter values. This dramatic increase causes $\hat{W}_i^l$ to become extremely large or small, risking overflow or instability. When lower exponent bits (e.g., b_3) or mantissa bits flip, the changes are much smaller and parameters stay within a manageable range. Our experiments of GPT-2 on SST-2 in Figure 2(c) show that flipping b_2 from 0 to 1 leads to the most significant performance degradation, while errors in lower bits have minimal impact. This demonstrates the importance of preserving the highest exponent bit for model stability and robustness.

4 Methodology

In this section, we introduce our defense strategies against bit-flip errors (BFEs) in large language models (LLMs), guided by the key observations from our analysis in Section 3. Our approach focuses on enhancing model robustness during the fine-tuning phase and mitigating the impact of BFEs in critical components of the model.

$\lvert W_i^l \rvert$ Range	Proportion	$\lvert W_i^l \rvert$ Range	Proportion
$(2^{-32}, 2^{-8}]$	3.106%	$(2^{-2}, 2^{-1}]$	6.183%
$(2^{-8}, 2^{-4}]$	37.09%	$(2^{-1}, 2]$	0.248%
$(2^{-4}, 2^{-2}]$	53.38%	$(2, +\infty)$	**0.0012%**

Table 2: Distribution of absolute parameter values for GPT-2 fine-tuned on the Wikitext-103 dataset. The proportion of $\lvert W_{i,j}^l \rvert$ greater than 2 is extremely small (1,445 of 124,439,808).

4.1 Defense Design

Self-Correction Fine-Tuning. According to Observation 3, BFEs during fine-tuning have a greater impact than those occurring during inference. This is because errors introduced during fine-tuning propagate into the learned parameters, leading to long-term degradation in the model. To address this, we propose a self-correction mechanism where the model is exposed to BFEs during fine-tuning, allowing it to learn to correct such errors.

We inject BFEs into the model during the fine-tuning phase. Let θ represent the original model parameters, and $\tilde{\theta}$ represent the parameters after introducing BFEs. For each parameter θ_i in layer l, a randomly selected bit position k in its floating-point representation by XOR operation:

$$\tilde{\theta}_i = \theta_i \oplus 2^k \tag{6}$$

This bit-flip injection allows the model to simulate errors that occur during deployment, forcing it to correct them during training. The total loss function $\mathcal{L}_{\text{total}}$ combines the standard task-specific loss $\mathcal{L}_{\text{task}}$ and a self-correction loss $\mathcal{L}_{\text{SC}}$, encouraging the model to learn outputs resilient to BFEs:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{\text{task}}(\theta) + \lambda_{\text{SC}} \mathcal{L}_{\text{SC}}(\tilde{\theta}, \theta) \\ \mathcal{L}_{\text{SC}}(\tilde{\theta}, \theta) &= \mathbb{E}_{x \sim \mathcal{D}} \left[\ell \left(f_{\tilde{\theta}}(x), f_{\theta}(x) \right) \right] \end{aligned} \tag{7}$$

where the self-correction loss $\mathcal{L}_{\text{SC}}$ is the divergence between outputs with and without BFEs. ℓ denotes l_2-norm by default.

Exponent Bit Protection. Observation 4 shows that BFEs in the highest exponent bit of floating-point parameters result in drastic performance degradation. To prevent significant magnitude shifts, we mask the highest exponent bit and further constrain parameter values during training. During inference, we modify the parameters to ensure the highest exponent bit remains 0. The corrected parameter $\hat{\theta}_i$ is then given by:

$$\hat{\theta}_i = \theta_i \,\&\, (1 \ll (N-2)) \tag{8}$$

Alternatively, to prevent parameters from reaching

Algorithm 1: FlipGuard

```
Input: Model parameters $\theta$, dataset $\mathcal{D}$,
       learning rate $\eta$, self-correction
       weight $\lambda_{\text{SC}}$, BFE rate $\rho$
for each epoch $e = 0, 1, \dots, E$ do
    Step 1: Randomly select $\rho \times |\theta|$
     parameters from $\theta$;
    for each selected parameter $\theta_i$ do
        Randomly select bit $k$ and flip:
         $\tilde{\theta}_i = \theta_i \oplus 2^k$;
    Step 2: Compute task loss:
     $\mathcal{L}_{\text{task}}(f_{\tilde{\theta}}(x), y)$;
    and self-correction loss:
     $\mathcal{L}_{\text{SC}} = \ell(f_{\tilde{\theta}}(x), f_\theta(x))$;
    $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \lambda_{\text{SC}} \mathcal{L}_{\text{SC}}$;
    Step 3: Update parameters by
     $\theta \leftarrow \theta - \eta \nabla \mathcal{L}_{\text{total}}$;
    Step 4: Apply exponent mask and clip:
     $\hat{\theta}_i = \text{clip}(\theta_i, -(2 - 2^{-M}), 2 - 2^{-M})$;
```

values that would set the highest exponent bit to 1, during fine-tuning, we clip the parameter values within a defined range $[-\epsilon, \epsilon]$, where $\epsilon = 2 - 2^{-M}$ and M denotes the number of mantissa bits, ensuring the highest exponent bit remains unaffected:

$$\hat{\theta}_i = \text{clip}(\theta_i, -\epsilon, \epsilon) \tag{9}$$

This range represents the safe interval within which the parameters can fluctuate without causing the most significant bit of the exponent to flip to 1. Thus, we avoid potential numerical instability or overflow issues that arise from range exceeding.

4.2 Discussion

Theoretical Analysis of BFEs and FlipGuard. Sensitivity analysis are widely used to study the robustness of neural networks by quantifying the relationship between input or parameter perturbations and output stability. Previous works (Virmaux and Scaman, 2018; Chen et al., 2024) show that Lipschitz continuity ensures bounded output changes for small input perturbations. Lipschitz constant is defined as $L = \sup_{x_1 \neq x_2} \frac{\|f(x_1) - f(x_2)\|}{\|x_1 - x_2\|}$, Such stable neural networks require small and bounded Lipschitz constant. However, bit-flip perturbations, especially in the highest exponent bit, can cause unbounded changes (e.g., up to 2^{128} for 32-bit FP value) which invalidates the utility of the Lipschitz constant. Our Exponent Bit Protection constrains parameters to a safe interval $[-\epsilon, \epsilon]$, ensuring that errors remain bounded. Besides, our Self-Correction Fine-Tuning trains the model to minimize divergence between outputs with and without BFEs, further bounding the Lipschitz constant.

Combination with LoRA and quantization. Quantization reduces model size and cost by operating at the weight and activation level, but BFEs still occur at the bit representation level in hardware. This distinction means BFEs can still occur post-quantization. While fewer bits in quantized models (e.g., 4-bit, 8-bit) reduce exponent-related perturbations, they may amplify BFE impact due to lower numerical precision. Our method, which self-corrects bit-level perturbations, remains fully compatible with other techniques. Combining it with LoRA (Hu et al., 2021) and quantization enhances computational efficiency while maintaining robustness to BFEs, enabling resource-efficient deployment in constrained environments. We further discuss computational overhead in 5.3.

5 Experiment

5.1 Experimental Setup

Models We use four widely-used pre-trained Large Language Models in our experiments: BERT (Devlin, 2018), GPT-2 Medium (Radford et al., 2019), OpenLlama-3B (Touvron et al., 2023; Geng and Liu, 2023), Gemma 2-2B (Team et al., 2024) and Llama 3.2-1B (Dubey et al., 2024).

Datasets. We employ eight datasets across various NLP tasks to evaluate the impact of bit-flip errors on model performance: MRPC (paraphrase detection) (Dolan and Brockett, 2005), MNLI (natural language inference) (Williams et al., 2017), SST-2 (sentiment classification) (Socher et al., 2013), CoLA (linguistic acceptability) (Warstadt, 2019), MMLU (multi-task language understanding) (Hendrycks et al., 2020), ARC-E (science question answering) (Clark et al., 2018), Wikitext-103 (language modeling, text generation) (Merity et al., 2016), SQuAD (question answering) (Rajpurkar, 2016), and GSM8K (math problem solving) (Cobbe et al., 2021).

Metrics. we report accuracy or F1-score for classification tasks and perplexity for generation tasks.

Implementation Details. All experiments were implemented using PyTorch, with models running on two RTX 2080Ti GPUs. We used pre-trained models from HuggingFace. Fine-tuning was conducted with a learning rate of 5e-5 and batch sizes of 16 (classification) and 8 (genera-

Model	Dataset	Clean	BFEs	SEC-DED	DHBFA	WRecon	RREC	NeuroPots	Ours
BERT	MRPC $\uparrow$	87.64$_{\pm 2.29}$	70.27$_{\pm 2.24}$	73.95$_{\pm 2.12}$	62.74$_{\pm 2.26}$	72.19$_{\pm 2.53}$	73.51$_{\pm 2.13}$	**76.18**$_{\pm 1.79}$	75.93$_{\pm 1.78}$
	MNLI $\uparrow$	84.32$_{\pm 0.74}$	68.73$_{\pm 0.95}$	71.92$_{\pm 0.81}$	66.89$_{\pm 0.99}$	63.27$_{\pm 1.35}$	69.45$_{\pm 0.66}$	72.16$_{\pm 0.85}$	**78.12**$_{\pm 0.79}$
	SST-2 $\uparrow$	93.18$_{\pm 1.52}$	76.12$_{\pm 1.71}$	78.56$_{\pm 1.48}$	70.19$_{\pm 1.67}$	78.84$_{\pm 1.59}$	77.63$_{\pm 1.70}$	79.27$_{\pm 1.63}$	**85.39**$_{\pm 1.54}$
	CoLA $\uparrow$	60.49$_{\pm 2.03}$	42.73$_{\pm 2.27}$	49.78$_{\pm 2.14}$	46.29$_{\pm 2.35}$	47.83$_{\pm 2.34}$	48.95$_{\pm 2.22}$	50.31$_{\pm 2.09}$	**52.67**$_{\pm 2.05}$
GPT-2	MMLU $\uparrow$	45.17$_{\pm 1.89}$	32.71$_{\pm 1.67}$	34.52$_{\pm 1.88}$	34.21$_{\pm 1.52}$	37.82$_{\pm 1.64}$	30.96$_{\pm 1.64}$	36.82$_{\pm 1.84}$	**38.94**$_{\pm 1.44}$
	SST-2 $\uparrow$	92.73$_{\pm 1.46}$	75.61$_{\pm 1.65}$	76.52$_{\pm 1.59}$	71.32$_{\pm 1.76}$	74.51$_{\pm 1.82}$	77.35$_{\pm 1.73}$	79.12$_{\pm 1.81}$	**80.58**$_{\pm 1.72}$
	ARC-E $\uparrow$	62.24$_{\pm 1.94}$	46.78$_{\pm 1.83}$	50.21$_{\pm 1.90}$	41.87$_{\pm 1.66}$	49.73$_{\pm 1.82}$	47.61$_{\pm 1.77}$	52.31$_{\pm 1.99}$	**54.12**$_{\pm 1.71}$
	Wikitext-103 $\downarrow$	40.45$_{\pm 2.87}$	92.64$_{\pm 3.56}$	88.45$_{\pm 3.03}$	78.34$_{\pm 3.44}$	83.83$_{\pm 3.55}$	97.39$_{\pm 3.83}$	75.72$_{\pm 3.25}$	**56.29**$_{\pm 3.11}$
Llama 1	MNLI $\uparrow$	83.21$_{\pm 0.72}$	67.12$_{\pm 0.97}$	70.56$_{\pm 0.87}$	68.29$_{\pm 0.96}$	71.11$_{\pm 1.30}$	65.79$_{\pm 0.93}$	73.18$_{\pm 0.99}$	**75.23**$_{\pm 0.73}$
	MMLU $\uparrow$	47.61$_{\pm 1.98}$	33.19$_{\pm 1.85}$	36.19$_{\pm 1.77}$	35.12$_{\pm 1.97}$	34.45$_{\pm 1.79}$	36.84$_{\pm 1.94}$	38.84$_{\pm 1.80}$	**39.34**$_{\pm 1.84}$
	ARC-E $\uparrow$	61.13$_{\pm 1.12}$	45.79$_{\pm 1.90}$	49.18$_{\pm 1.14}$	48.26$_{\pm 1.52}$	50.31$_{\pm 1.34}$	47.23$_{\pm 2.20}$	52.49$_{\pm 1.01}$	**53.94**$_{\pm 1.15}$
	Wikitext-103 $\downarrow$	22.89$_{\pm 2.20}$	55.94$_{\pm 3.09}$	43.61$_{\pm 2.53}$	49.82$_{\pm 2.8}$	55.23$_{\pm 2.60}$	52.34$_{\pm 3.20}$	44.91$_{\pm 2.50}$	**40.62**$_{\pm 2.20}$
Gemma 2	MRPC $\uparrow$	88.68$_{\pm 2.32}$	69.38$_{\pm 2.17}$	72.14$_{\pm 2.08}$	68.15$_{\pm 2.35}$	70.92$_{\pm 2.22}$	67.81$_{\pm 2.29}$	**75.29**$_{\pm 2.24}$	73.88$_{\pm 1.81}$
	SST-2 $\uparrow$	91.23$_{\pm 1.89}$	74.58$_{\pm 1.98}$	76.79$_{\pm 1.95}$	75.23$_{\pm 1.95}$	72.84$_{\pm 1.81}$	77.64$_{\pm 1.96}$	**80.17**$_{\pm 1.87}$	79.64$_{\pm 1.77}$
	ARC-E $\uparrow$	63.87$_{\pm 1.91}$	47.26$_{\pm 2.02}$	51.74$_{\pm 1.77}$	50.91$_{\pm 1.92}$	48.12$_{\pm 1.75}$	51.34$_{\pm 2.00}$	54.32$_{\pm 1.74}$	**55.63**$_{\pm 1.69}$
	Wikitext-103 $\downarrow$	23.45$_{\pm 2.90}$	60.64$_{\pm 3.24}$	58.45$_{\pm 2.88}$	50.34$_{\pm 3.09}$	54.83$_{\pm 3.23}$	59.39$_{\pm 3.55}$	48.72$_{\pm 2.14}$	**48.29**$_{\pm 2.94}$
Llama 3.2	SQuAD $\uparrow$	86.17$_{\pm 1.12}$	67.84$_{\pm 1.31}$	72.91$_{\pm 1.25}$	69.83$_{\pm 1.20}$	74.23$_{\pm 1.16}$	75.92$_{\pm 1.29}$	74.15$_{\pm 1.28}$	**78.35**$_{\pm 1.05}$
	GSM8K $\uparrow$	44.41$_{\pm 1.48}$	32.35$_{\pm 0.98}$	35.63$_{\pm 1.70}$	36.81$_{\pm 1.05}$	34.95$_{\pm 1.62}$	32.53$_{\pm 0.71}$	37.84$_{\pm 2.62}$	**38.84**$_{\pm 1.50}$

Table 3: Performance comparison of models on corresponding datasets under various inference conditions: clean, with 1% BFEs, and using different defense methods. Accuracy is the evaluation metric for tasks {MRPC, MNLI, SST-2, CoLA, MMLU, ARC-E, GSM8K}, perplexity for task {Wikitext-103}, and F1-score for task {SQuAD}.

tion), across 5 epochs. Bit-flip errors were simulated at various rates. Specifically, we simulate errors at 0.1%, 1%, 5%, and 10% of the model's numerical representations during fine-tuning and inference by default. The code can be found at `https://github.com/UNITES-Lab/FlipGuard`.

Counterparts. To validate the effectiveness of our proposed defense strategies, we compare them against several baseline and state-of-the-art methods designed to mitigate bit-flip errors:

- **SEC-DED** (Hamming, 1950): Single Error Correction, Double Error Detection, a classic error correction technique for hardware fault tolerance.
- **DHBFA** (He et al., 2020): A defense against adversarial BFAs by leveraging binarization-aware training and piece-wise clustering.
- **WRecon** (Li et al., 2020): A method focused on recovering corrupted weights during inference by reconstructing weights affected by bit-flip errors.
- **RREC** (Liu et al., 2022): Uses redundant error-correcting codes to prevent BFEs propagation.
- **NeuroPots** (Liu et al., 2023): Introduce honey neurons to attract and trap bit-flips.

5.2 Comparisons to State-of-the-Art

We compare accuracy and perplexity across all methods on clean data and under a 1% BFE rate, as shown in Table 3.

❶ Impact of BFEs on Performance. BFEs degrade performance significantly across all models. For instance, BERT accuracy on MRPC drops from 87.64% to 70.27%, while Gemma 2's perplexity on Wikitext-103 rises sharply from 23.45 to 60.64, highlighting the severity of BFE-induced errors.

❷ Effectiveness of Defense Mechanisms. Defense methods mitigate BFE impact to varying extents. For BERT on MRPC, our method improves accuracy from 70.27% to 75.93%, outperforming SEC-DED (73.95%) and approaching NeuroPots (76.18%). For Llama 1-3B on Wikitext-103, our defense reduces perplexity from 55.94 to 40.62, demonstrating strong error resilience.

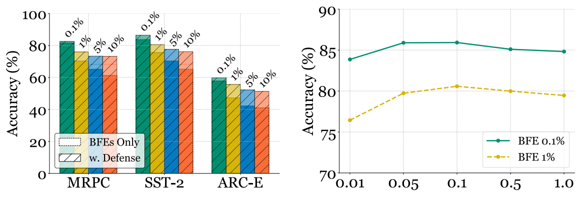
❸ Comparison of Defense Methods. Across models, our defense consistently achieves superior results. For instance, in Llama 1-3B on MNLI, accuracy improves from 67.12% to 75.23%, surpassing other defenses and confirming its robustness against BFEs.

5.3 Diagnostic Analysis

Ablation Study on Defense Components. We evaluate the impact of the two defense mechanisms through four configurations: (1) baseline (no defense), (2) self-correction fine-tuning, (3) exponent bit protection, and (4) their combination. Table 4 presents the results across BFE rates {0.1%, 1%,

SC	EP	0.1%	1%	5%	10%
✗	✗	81.31	75.61	66.31	60.81
✓	✗	$82.44_{\uparrow 1.13}$	$77.32_{\uparrow 1.71}$	$72.12_{\uparrow 5.81}$	$70.97_{\uparrow 10.16}$
✗	✓	$84.32_{\uparrow 3.01}$	$78.34_{\uparrow 2.73}$	$74.23_{\uparrow 7.92}$	$71.56_{\uparrow 10.75}$
✓	✓	**85.92**$_{\uparrow 4.61}$	**80.58**$_{\uparrow 4.97}$	**76.56**$_{\uparrow 10.25}$	**74.12**$_{\uparrow 13.31}$

Table 4: **Ablation Study** on defense components under varying BFE rates for GPT-2 on the SST-2 dataset. BFE rates are set to {0.1%, 1%, 5%, 10%}. SC denotes Self-Correcting Fine-tuning and EP denotes Exponent Bit Protection in Section 4.



(a) Different BFEs rates. (b) λ_{SC} Sensitivity Analysis.

Figure 3: (a) Performance of models under different bit-flip error rates {0.1%, 1%, 5%, and 10%}. We test BERT on MRPC, GPT-2 on SST-2, and Gemma 2 on ARC-E. (b) Sensitivity analysis assessing the impact of varying λ_{SC} on model robustness and task accuracy.

5%, 10%}. While both defenses individually improve performance, the combined strategy consistently outperforms them, especially at higher BFE rates. For instance, at 10% BFEs, the combined defense achieves the highest accuracy of 74.12%.

Resilience to BFEs at Different Error Rates. We evaluate model resilience to varying BFE rates ({0.1%, 1%, 5%, 10%}) on BERT (MRPC), GPT-2 (SST-2), and Gemma2 (ARC-E). Without defense, accuracy steadily drops, while our defense mechanisms significantly mitigate performance degradation, especially at higher BFE rates.

Hyper-parameter λ_{SC} Sensitivity Analysis. We analyze the effect of the self-correction weight λ_{SC} on model robustness. As shown in Figure 3, increasing λ_{SC} improves resilience to BFEs, but excessively high values risk overfitting to errors, reducing overall performance.

Defense for Component-Specific BFEs. Using GPT-2 on MMLU, we inject 1% BFEs into individual components (embedding, self-attention, MLP, LayerNorm) and apply our defense strategies. Results show our methods are effective across all components, with embedding and self-attention layers benefiting the most. Even for less vulnerable components like LayerNorm, our defenses improve stability, demonstrating robustness at both whole-network and component levels.

Impact of Fixed BFEs *Count*. Using the same BFE *rate* across modules allows us to compare

Component	w/o. Defense	w. Defense
Embedding	29.10	**37.25**$_{\uparrow 8.15}$
Self-Attention	31.32	**38.45**$_{\uparrow 7.13}$
MLP	39.20	**39.90**$_{\uparrow 0.70}$
LayerNorm	43.10	**43.50**$_{\uparrow 0.40}$

Table 5: GPT-2 accuracy on the MMLU dataset under component-specific BFEs (1%), comparing performance with and without the proposed defense strategies.

the relative sensitivity of different components, as it simulates real-world hardware faults where all parameters have the same probability of bit-flips. We further analyze the impact of fixed BFE *count*. Despite not having the largest parameter size, the embedding layer exhibits the most significant performance degradation, highlighting its vulnerability. Table 6 confirms that the embedding layer remains the most sensitive.

Component	1 Bit	10 Bits	100 Bits	1000 Bits
Embedding	92.67	92.02	90.56	88.34
Attention	92.73	92.12	91.67	90.45
MLP	92.73	92.52	92.40	92.23
LayerNorm	92.73	92.68	91.65	89.43

Table 6: Impact of fixed BFE count on GPT-2 model performance on the SST-2 dataset.

Computational Overhead. The primary computational overhead of FlipGuard comes from computing the self-correction loss, which requires outputs from both clean and perturbed parameters. This overhead scales linearly with model size. However, leveraging LoRA (Hu et al., 2021) reduces this cost by enabling clean and perturbed outputs in a single forward pass, minimizing matrix multiplication overhead. Although separate activation function applications (e.g., Softmax) are needed, their cost is negligible compared to the linear transformations. On SST-2 with GPT-2, FlipGuard with LoRA adds only 22% computational time compared to standard fine-tuning but improves performance by 4.97%.

6 Conclusion

In this paper, we systematically evaluated the effects of bit-flip errors (BFEs) across multiple levels of large language models (LLMs), including model parameters, activations, gradients, and bit positions. Our simulations revealed distinct vulnerabilities in different components. To mitigate the impact of BFEs, we introduced a defense strategy that significantly improves LLM robustness. Our findings highlight the need for robust error-mitigation techniques to ensure the reliability of LLMs across diverse deployment environments.

Acknowledgement. This work is partially supported by Amazon Research Award, Cisco Faculty Award, UNC Accelerating AI Awards, NAIRR Pilot Award, OpenAI Researcher Access Award, and Gemma Academic Program GCP Credit Award.

Limitation Discussions & Future Work

While our study provides comprehensive insights into the impact of bit-flip errors on large language models and proposes effective defense mechanisms, several limitations remain that open avenues for future research.

First, our experiments primarily focus on popular models, including BERT, GPT-2, Gemma, and Llama. While these models represent diverse transformer architectures, LLMs vary significantly in their scale, training regimes, and specific optimizations. Future work could extend our analysis to other LLMs, such as newer models like GPT-3, to verify the generalizability of our findings and defense mechanisms across different architectures and parameter scales.

Second, We evaluated our defense mechanisms under specific BFE rates {0.1%, 1%, 5%, and 10%}. However, real-world hardware-induced errors vary dynamically based on hardware age, workload, and environment. Future studies should simulate these dynamic conditions for a more comprehensive understanding of model behavior and defense robustness.

Third, our current work focuses on post-training defense mechanisms. Investigating the application of these methods during training could provide valuable insights into how training-time error simulations influence model robustness. While retraining large-scale LLMs from scratch remains computationally prohibitive for us, future work by organizations with access to extensive computational resources could explore this avenue. Simulating bit-flip errors during training might act as a form of regularization, akin to dropout, potentially enhancing the resilience of models under perturbed conditions.

Lastly, the proposed defense mechanisms, especially during fine-tuning, introduce computational overhead. This may limit practicality for time-sensitive applications. Optimizing these mechanisms to reduce runtime costs, possibly through lightweight quantization or hardware-level support, remains an important area for future research.

Ethical Statement

Our research focuses on defending Large Language Models (LLMs) against bit-flip errors (BFEs) to enhance AI system reliability and security. While our findings could be misused to exploit hardware vulnerabilities, we present our methods responsibly, emphasizing countermeasures rather than attack details. All experiments were conducted in controlled environments, without real user data and adhering strictly to ethical guidelines to ensure that our work supports the development of secure and trustworthy AI technologies.

References

Pieter Abbeel Ajay Jain, Tianjun Zhang. 2024. Towards robust and scalable large language models. *EECS, University of California, Berkeley*.

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

Erh-Chung Chen, Pin-Yu Chen, I Chung, Che-Rung Lee, et al. 2024. Data-driven lipschitz continuity: A cost-effective approach to improve adversarial robustness. *arXiv preprint arXiv:2406.19622*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third international workshop on paraphrasing (IWP2005)*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Xinyang Geng and Hao Liu. 2023. Openllama: An open reproduction of llama. *URL: https://github.com/openlm-research/open_llama*.

Richard W Hamming. 1950. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160.

Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. 2020. 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.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. 2024. Zenhammer: Rowhammer attacks on amd zen-based platforms. In *33rd USENIX Security Symposium (USENIX Security 2024)*.

Xun Jiao, Fred Lin, Harish D Dixit, Joel Coburn, Abhinav Pandey, Han Wang, Jianyu Huang, Venkat Ramesh, Wang Xu, Daniel Moore, et al. 2024. Pvf (parameter vulnerability factor): A quantitative metric measuring ai vulnerability and resilience against parameter corruptions. *arXiv preprint arXiv:2405.01741*.

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. *ACM SIGARCH Computer Architecture News*, 42(3):361–372.

Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. 2020. Rambleed: Reading bits in memory without accessing them. In *41st IEEE Symposium on Security and Privacy (S&P)*.

Salah Lhoussaine, Georgios Tziantzioulis, Michael B. Sullivan, Vilas Sridharan, Nathan DeBardeleben, Christian Engelmann, and Simranjit Singh. 2024. A first look at bfloat16 soft-error resilience in large language models. *Preprint*, arXiv:2412.07192.

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

Liang Liu, Yanan Guo, Yueqiang Cheng, Youtao Zhang, and Jun Yang. 2022. Generating robust dnn with resistance to bit-flip based adversarial weight attack. *IEEE Transactions on Computers*, 72(2):401–413.

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)*, pages 6347–6364.

Ziyao Liu, Jinyuan Jia, Jiachen T. Wang, Wenbo Guo, Zhaofeng He, and Xinyang Zhang. 2024. Bit-sponge: A bit-level attack and defense for large language models. *Preprint*, arXiv:2411.13757.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Shubhendu S Mukherjee, Christopher Weaver, Joel Emer, Steven K Reinhardt, and Todd Austin. 2003. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 29–40. IEEE.

Alec Radford. 2018. Improving language understanding by generative pre-training.

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

P Rajpurkar. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Aladin Virmaux and Kevin Scaman. 2018. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31.

A Warstadt. 2019. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2017. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. 2016. One bit flips, one cloud flops:{Cross-VM} row hammer attacks and privilege escalation. In *25th USENIX security symposium (USENIX Security 16)*, pages 19–35.