FT2: First-Token-Inspired Online Fault Tolerance on Critical Layers for Generative Large Language Models

Yu Sun[§], Zhu Zhu[§], Cherish Mulpuru[§], Roberto Gioiosa[†], Zhao Zhang[‡], Bo Fang[†*], and Lishan Yang[§]
[§] *Department of Computer Science, George Mason University, USA, {ysun23,zzhu22,cmulpuru,lyang28}@gmu.edu*
[†] *Pacific Northwest National Lab, USA, {roberto.gioiosa,bo.fang}@pnnl.gov*
[‡] *Department of Electrical and Computer Engineering, Rutgers University, USA, zhao.zhang@rutgers.edu*

ABSTRACT

Generative Large Language Models (LLMs) are deployed on large-scale computing systems, where such tasks unavoidably suffer from soft errors, leading to quality degradation of content generated by LLMs. Enhancing LLM resilience is particularly challenging because of its complicated model architecture and tremendous size. State-of-the-art protections have limitations such as high overhead and incomplete coverage, and often require offline profiling.

In this work, we design FT2, a First-Token-inspired online Fault Tolerance methodology for generative LLMs, offering high reliability and low overhead. Inspired by our comprehensive characterization study, FT2 selectively protects critical layers on the fly during inference and leverages the information (bounds) from the first token generation to protect the generation of the following tokens based on the similarity of the input data during token generation. We extensively evaluate FT2 across seven LLMs and three datasets under three fault models, achieving 92.92% silent data corruption (SDC) rate reduction with only 3.42% overhead on average.

CCS CONCEPTS

• **Computer systems organization** → **Reliability**; • **Computing methodologies** → **Natural language processing**.

KEYWORDS

Reliability, Large Language Models (LLMs), Resilience Enhancement

ACM Reference Format:
Yu Sun[§], Zhu Zhu[§], Cherish Mulpuru[§], Roberto Gioiosa[†], Zhao Zhang[‡], Bo Fang[†*], and Lishan Yang[§] . 2025. FT2: First-Token-Inspired Online Fault Tolerance on Critical Layers for Generative Large Language Models. In *The 34th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '25), July 20–23, 2025, Notre Dame, IN, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3731545.3731570

* Corresponding author.

HPDC '25, July 20–23, 2025, Notre Dame, IN, USA

ACM ISBN 979-8-4007-1869-4/2025/07.
https://doi.org/10.1145/3731545.3731570

1 INTRODUCTION

Deep learning models, especially generative Large Language Models (LLMs), are widely used for various tasks such as text generation, machine translation, question answering, and code generation [1, 48, 63, 66]. The growing size and complexity of LLMs enhance their ability to solve increasingly complex and challenging problems, requiring substantial computational resources. As a result, the intensive inference tasks are preferably hosted by large-scale computing systems, where such tasks inevitably face the challenges of increased probability of encountering soft errors (i.e., transient hardware faults) [51–53, 65]. As the most commonly observed errors in computing systems, these soft errors originate from cosmic radiation [21], shrinking transistors [8], and low-voltage operations [10], resulting in serious outcomes such as silent data corruptions (SDCs), e.g., wrong answers or misleading information in the context of LLM inference. Therefore, mitigating the impact of such errors on LLM inference is of great importance.

Limitations of state-of-the-art approaches. Depending on the types of faults encountered in large-scale systems, existing fault tolerance techniques have fundamental limitations. For instance, faults affecting the computational path would bypass the ECC (Error Correction Code) in memory and propagate during execution. On the other hand, common software-based approaches are typically designed for fail-stop failures (i.e. checkpoint/restart) or potentially incur non-trivial overhead (i.e. algorithm-based checksum). Among those, range restriction is a particularly useful technique due to its negligible overhead (less than 1% for convolutional neural networks) [12, 58] and high effectiveness. Unlike other techniques that are applied throughout LLM execution, range restriction protects [1] LLM inference at a coarser granularity after layer execution. Range restriction achieves protection by checking the neuron values of each protected layer and clipping outlier neurons to 0 (or a small value) according to the bounds obtained by offline profiling using the training datasets, to eliminate the impact of extremely corrupted values on model output.

However, existing range-restriction-based solutions [12] have two major limitations. First, the SDC rate remains considerable (an average of 2.08% for the models and datasets studied in this work) even with existing solutions applied. Through our quantitative characterization study, we identify the underlying reason: the lack of fundamental approaches for identifying *critical layers*, i.e., ignoring the protection of these layers results in non-negligible SDC outcomes. Second, all range-restriction techniques rely on offline profiling to obtain the bounds of neurons. Such processes, however, are not always feasible for LLMs, as the training datasets

[1] Note that in this paper we refer *protection* as performing both error detection and correction to prevent LLM inference from generating silent data corruptions.

may be unavailable for LLM tasks [24, 41, 69]. Offline bound profiling also suffers from high profiling costs (4.7 - 217.5 hours on an NVIDIA A100 GPU, as shown in Section 3).

Motivation and Challenges. The aforementioned limitations of range-restriction techniques motivate us to design a practical, automated LLM fault tolerance solution that effectively explains and leverages LLM characteristics. However, it is complicated to analyze and pinpoint the critical components in LLMs due to their growing size and complexity. Additionally, considering the limited input data per LLM inference, designing an online-only technique with effective bounds and high fault tolerance is particularly challenging.

Key insights and contributions. To overcome the challenges, we design and present FT2, a First-Token-inspired online Fault Tolerance methodology on critical layers for generative LLMs, offering high reliability and low overhead. Through extensive fault injection experiments, we assess the criticality of layers, i.e., the necessity of protecting certain layers. Our quantitative analysis demonstrates that protecting the identified critical layers brings huge reliability benefits. We further analyze the LLM model architecture and reveal the underlying reasons for layer criticality: residual branches can recover the information loss due to NaN, while scaling operations and activation layers can reduce the magnitude of faulty values and change the NaN distribution under the impact of hardware faults. We propose a *heuristic* to classify critical/non-critical layers: a layer is critical if no scaling operation or activation layer is present before the following linear layer.

When facing generative LLMs, the input for generating the first token is a subset of the input for generating the following tokens, therefore the neuron values have similarities. This enables another *key idea* of online FT2: leveraging the information of the first token generation to infer the bounds of generating other tokens. This idea is supported by two key insights derived from our study: the bounds are reusable between the first token and the following tokens and the impact of soft errors on the first token generation is marginal (detailed in Section 4). These two key insights are validated through progressive reasoning and experiments. For error correction, we pinpoint a unique characteristic of generative LLMs, the existence of large neuron values in generative LLMs. This unique characteristic guides our design choice of clipping extreme out-of-bound values inside the bound.

Our main contributions are summarized as follows:

- We perform a thorough and detailed characterization study to assess LLM layer criticality and explain the underlying reasons. We propose an efficient heuristic to classify critical/non-critical layers: a layer is critical if no scaling operation or activation layer is present before the next linear layer.
- We identify a unique feature of generative LLM tasks, the input similarity among the generation of tokens. FT2 leverages this unique feature to achieve online-only protection: bounds are profiled during the first token generation and then used to protect the generation of the following tokens.
- We present FT2, a First-Token-inspired online Fault Tolerance methodology for generative LLMs, offering high reliability and low overhead. We pinpoint a unique characteristic of generative LLMs, the existence of large neuron values (see subsection 4.3 for details), which informs our design choice of clipping extreme out-of-bound values to the bound.
- We extensively evaluate FT2 across seven LLMs and three datasets under three fault models to show their effectiveness and efficiency with over 11 million fault injection experiments. We also demonstrate FT2 with data types and hardware configurations. FT2 outperforms the existing protection mechanisms and achieves 92.92% SDC rate reduction with only 3.42% overhead on average.

Experimental methodology and artifact availability. We evaluate our methodology on two sets of NVIDIA GPUs, A100 and H100. We consider common fault models used in reliability studies [12, 18, 44, 72], including single-bit flip, double-bit flip, and single-bit flip in Exponent bits. The fault injection experiments are performed using the hook mechanism of PyTorch, which is similar to PyTorchFI [45]. We choose the state-of-the-art range-restriction-based solutions: Ranger [12], MaxiMals [57], and Global Clipper [60] as baselines. We have publicly released FT2 and its associated fault injection framework[2].

Limitations of the proposed approach. We recognize that FT2 uses heuristics to target critical layers, and speculates the bounds obtained from the first token for the following tokens. Our heuristics are under the assumptions that no training data is available for offline bound learning, and the resource requirements of applying such a range restriction to all layers are extensive. In extreme cases where the safety-critical applications are present, achieving 0% SDC may require additional techniques such as duplications in place, where the corresponding significant overhead is expected.

2 BACKGROUND

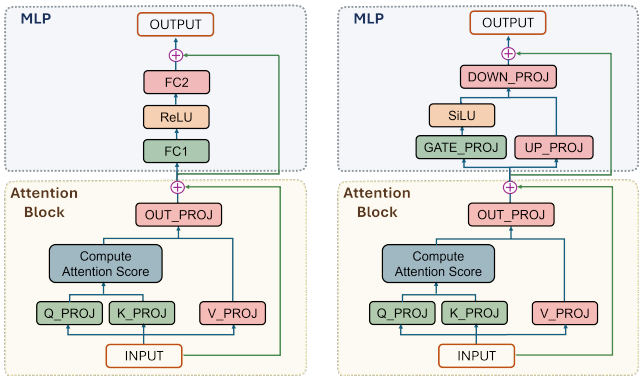
In this section, we present an overview of Large Language Models. Then, we describe the fault models and fault injection techniques that are extensively employed in this study.

2.1 Large Language Models

Large Language Models (LLMs) have emerged as a groundbreaking advancement in artificial intelligence, particularly in natural language processing. Most LLMs are transformer-based models, consisting of encoders and/or decoders that rely on the self-attention mechanism. The encoder extracts features from the input, and the decoder produces different outputs for different downstream tasks based on these features. Typically, generative LLMs are decoder-only models, where the decoders can attend to (i.e., access information from) previous tokens only through causal masking, further improving their capabilities for generation.

Most LLMs have similar architectures that consist of a cascade of transformer blocks. Each block has two major components: an attention block and a Multi-Layer Perceptron (MLP) (Figure 1). In the attention block, the input is transformed into Key (K), Query (Q), and Value (V) through their respective projection layers. The attention weights, computed from Q-K interactions, are applied to V. The weighted output then undergoes a final linear transformation through the out_proj layer. The Multi-Layer Perceptron (MLP) consists of two or three linear layers and one activation layer, following the attention block. This architecture aims to enhance the model

[2] https://github.com/pipijing13/FT2-LLM-inference-protection



(a) OPT-6.7B (b) Llama2-7B

Figure 1: Different LLM Model Architectures.

representation capability after the attention mechanism. The LLMs considered in this work have different linear layer settings. Here we list two examples, OPT-6.7B in Figure 1(a) and Llama2-7B in Figure 1(b). The main differences lie in the number and arrangement of linear layers in MLP.

2.2 Fault Model

The soft errors are the manifestation of hardware transient faults originating from sources like cosmic radiation [21], shrinking transistors [8], and low voltage operations [10]. In this study, we assume that register files, caches, and memory are protected by Error Correction Code (ECC), which is the case in modern CPUs and GPUs used for LLM tasks [23]. Hence, the fault model considered is the computation-related faults affecting computational hardware components such as ALUs (Arithmetic Logic Units). This fault model aligns with the prior studies [5, 22, 36]. In particular, we consider three fault types: 1) *1-bit:* single-bit flip, 2) *2-bit:* double-bit flip, and 3) *EXP:* single-bit flip in Exponent bits [26], representing the typical faults affecting the computation. We consider the EXP fault model as a more challenging scenario since bit flips in exponent bits are more likely to have severe consequences [26, 55].

2.3 Fault Injection

To understand the impact of hardware faults on the LLM computations, we rely on fault injection (FI) experiments. A fault injection experiment in our study is to mimic the behavior of a fault (e.g. single- or multi-bit flips) occurring during the computation systematically and observe what is the outcome of the fault [12, 35, 40].

Assumptions. We only consider soft errors during the inference phase of the LLMs, since typically the LLMs are trained once and repeatedly used by a large number of users [1, 11, 29, 63]. We assume that only one error occurs per inference because it is unlikely that multiple faults occur during the short period of a single inference (usually in milliseconds or seconds). Our assumption aligns with the prior studies [2, 5, 20, 35, 36, 56, 59, 68].

Tool. A couple of fault injection frameworks for convolutional neural networks are publicly available, such as TensorFI [13], PyTorchFI [46], and SNIFF [9]. Our fault injection extends the hook mechanism of PyTorch to directly inject faults into randomly selected neurons, which is similar to PyTorchFI. For each fault injection trial, the location of the model to inject a fault is identified by the layer ID, neuron ID, and bit locations. Since LLMs primarily perform matrix computations associated with neurons [31], our fault injection campaign covers the predominant hardware operations.

Outcome category. The outcome of a fault injection experiment can be categorized as follows:

- **Masked:** The injected fault does not affect the correctness of LLM outputs. There are two types of masked output for LLMs: (1) the text generated by LLM is identical to the one from fault-free execution; (2) the output is different from the fault-free one but the generated text is semantically correct, i.e., if the meaning of the answer is equivalent to the reference answer (i.e., the information asked by the question). For example, "The number of people is 5" is also correct compared to the fault-free answer "There are 5 people" for question-answering tasks; thus, it is considered a masked outcome.
- **Silent Data Corruption (SDC):** The text generated by LLM is different from the fault-free execution and is semantically wrong. For instance, "There are 4 people" should be an SDC compared with the fault-free output "There are 5 people". If the answer does not contain or partially contains the reference answer, it is classified as a wrong answer and thus treated as an SDC outcome.

Note that the outcome of *one* fault injection run only denotes the resilience of that location of the fault injection, i.e., the vulnerability of that specific fault site to bit flips. To evaluate the resilience of the whole LLM, one should perform exhaustive fault injection experiments at *every* fault site. However, covering all fault sites is impossible, as the total number can reach billions or even trillions for LLMs. Instead, we conduct a statistical fault injection, aligned with prior works [2, 73–75]. Each experiment involves injecting a single fault at a random (i.e., uniformly distributed) location. We only consider linear layers in the decoder blocks of the model, since they consist of most of the computation (e.g., 94% in Llama2-7B model with sequence length set to 1024) and thus are more likely to encounter soft errors. In total, we have conducted over 11 million FI experiments, with around 8000 GPU hours.

3 MOTIVATIONAL STUDY

In this section, we pinpoint the limitations of the state-of-the-art range-restriction-based solutions and quantitatively measure their deficiencies. We start with their deficiency in protecting LLMs, then show the undesirable high cost of profiling. Among various protection techniques, range restriction is particularly popular due to its negligible protection overhead compared to other protection techniques [12, 58]. Range restriction protects neural networks (NNs) by clipping outlier neurons to 0 (or a small value) according to the bounds obtained by offline profiling using the training datasets, to eliminate the impact of extreme corrupted values on model output. Range restriction is originally designed for Convolutional NNs (CNNs) and there are several works applying it to the transformer-based models. For example, MaxiMals [57] protects the MLP layers instead of only protecting the activation layers which is the choice of the original strategy [12]. Global Clipper [60] partially protects the linear layers in the attention block and corrects the

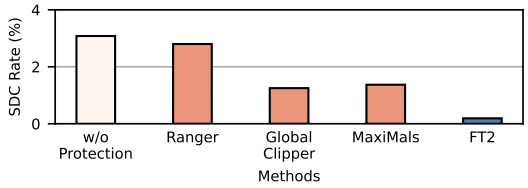


Figure 2: LLM Resilience with various protections applied on Llama2-7B models using GSM8K dataset. A considerable number of SDCs still exist after applying the existing range-restriction-based methods. On the other side, our FT2 provides much more effective protection.

Not-a-Number (NaN) values. However, these methods focus on vision transformers, raising the issue of applying them to generative LLMs, as we show below.

3.1 Limitation 1: SDC Rate Reduction

We first present the limitation observed from the existing range-resriction-based methods for the SDC rate reduction capability. As a motivating example, Figure 2 shows the SDC rate with and without various existing range-clipping-based protection methods, obtained through FI experiments under the EXP fault model, with GSM8K dataset [15] and Llama2-7B model [66]. Compared to the results on models without protection mechanisms applied, Ranger and MaxiMals only reduce the SDC rate by 0.28% and 1.71%, respectively. The best is Global Clipper with a 1.25% SDC rate after protection, which is still not negligible. In contrast, in this paper we show that our method FT2 achieves a much lower SDC rate of 0.19% in this case.

Motivation #1: Existing range-restriction-based protection methods cannot provide sufficient protection. This motivates us to perform in-depth analysis and design strong protection mechanisms to achieve low SDC rate.

3.2 Limitation 2: Bound Profiling

All existing range-restriction-based protection methods require offline profiling before their online application. Reportedly, this process uses 20% of the training dataset [12, 60]. These data are fed into the ML model to perform a forward pass to obtain the lower and upper bounds of each protected layer, which are then employed to clip the out-of-bound neurons during inference. This offline bound profiling is acceptable for CNN-based vision tasks such as image classification and object detection where the profiling datasets are publicly available, to name a few, ImageNet [16] and COCO [39]. Meanwhile, the size of the vision model is relatively small, allowing offline profiling that takes less than an hour [12]. However, for LLMs with especially generative tasks, there are two main issues of this offline profiling phase: 1) the lack of datasets and 2) high profiling costs.

Lack of datasets. The lack of datasets for various LLM tasks is a well-acknowledged problem [24, 41, 69]. Considering the real-world deployment of LLMs, the dataset that can be exploited for bound profiling requires representative user prompts (i.e., user inputs).

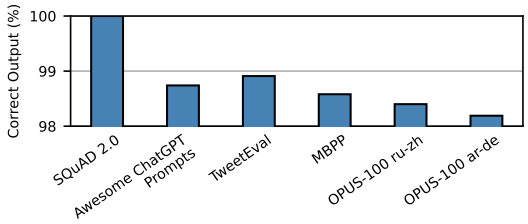


Figure 3: The percentage of correct outputs of OPT-6.7B model on SQuAD 2.0 question-answering dataset when applying protection using bounds from various datasets. Applying the alternative datasets decreases performance. Note that no fault injection is performed in this experiment.

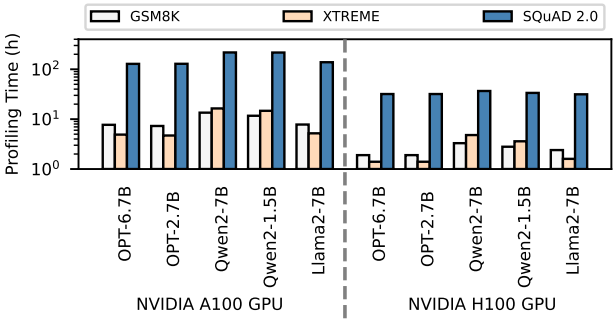


Figure 4: The bound profiling time for different tasks. Note that the y-axis is in log scale.

To compensate for the lack of datasets for bound profiling, an alternative solution is to use the bounds obtained from other similar datasets. Here we show that using the alternative datasets causes the decrease of the correct output percentage, i.e., the quality of the generated text is weaker. In this example, we use SQuAD 2.0 dataset [54] as the target dataset of LLM inference and we select 26,000 inputs (20% of the training dataset) to obtain the bounds. We randomly select another 50 inputs for inference with protection applied but no fault injected. We use Awesome ChatGPT Prompts [3], TweetEval [7, 49], MBPP [6], and OPUS-100 [64, 77] as the alternative datasets to obtain bounds, mimicking the case where SQuAD 2.0 dataset being not available. Figure 3 shows the percentage of correct outputs when applying these bounds to protect OPT-6.7B model in fault-free inference. Since the inputs selected in our study only involve the correctly answered inputs, the percentage of correct outputs is 100% for fault-free execution without applying any protection (the leftmost bar). When using bounds profiled from other datasets, the percentage of correct output drops by 1.09% to 1.81% in fault-free scenarios. Compared to the SDC rate of generative LLM inference under the single-bit fault model, which generally ranges from 0.92% to 3.03% without protection, this decrease raises a significant concern.

The observed degradation in generated text quality due to the use of alternative datasets essentially reflects the data dependency of the bounds. Since input content can vary significantly across datasets, especially those designed for different tasks, the diversity of input

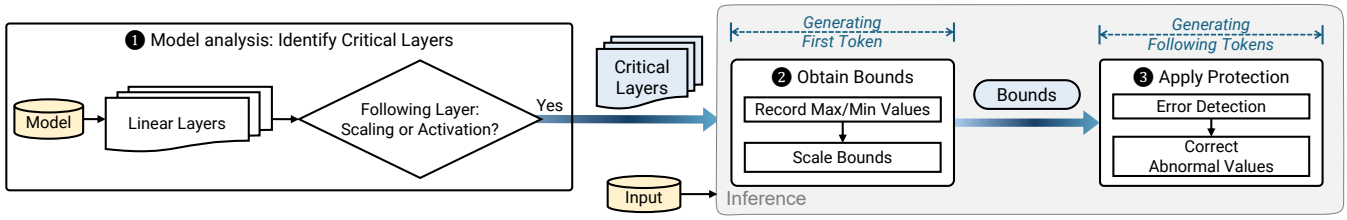


Figure 5: Overview of FT2. Firstly, critical layers are identified through model analysis. During the LLM inference, bounds are obtained when generating the first token and applied for the protection of generation following tokens.

tokens leads to significant differences in the bounds of neurons. Therefore, range restriction based on the alternative datasets may create false positives to convert benign neurons into incorrect ones without the presence of faults.

High Profiling Cost. Even if the dataset is available, the bound profiling cost can be unacceptable due to the large model sizes and the long output length for each inference instance. Existing solutions use 20% (Ranger and Global Clipper) of the training dataset or the whole validation dataset (MaxiMals). Figure 4 shows the profiling time, which can reach 217.5 hours using an NVIDIA A100 GPU with 20% training data. The profiling process still takes up to 36.7 hours with a more powerful NVIDIA H100 GPU. As the LLM size and the input length increase [30, 62], this profiling time would become essentially unacceptable in the future.

Motivation #2: The mandatory bound profiling is expensive, sometimes even unattainable due to dataset unavailability. This motivates us to design new techniques that obtain bounds without high-cost offline profiling.

4 METHODOLOGY

In this section, we present FT2, a First-Token-inspired online Fault Tolerance methodology for generative LLMs, offering high reliability and low overhead by selectively protecting critical layers. The workflow of FT2 is shown in Figure 5 and briefly described below:

- The first step is to identify critical layers given the model architecture of an LLM. From our in-depth reliability characterization (subsection 4.1), we derive a key insight for critical layer identification: a layer is deemed critical (i.e., necessary to be protected) if no scaling operation or activation layer is present between itself and the next linear layer.
- The second step is to obtain bounds while generating the first token of LLM inference. This is based on our second key insight (subsection 4.2): the bounds profiled during the first token generation are efficient for detecting the errors during the rest token generation.
- The third step is the online protection by applying range restrictions on critical layers during the token generation. We selectively protect the critical layers to minimize the runtime overhead of protection. An LLM-unique characteristic in neuron values is employed to improve the error correction accuracy (subsection 4.3).

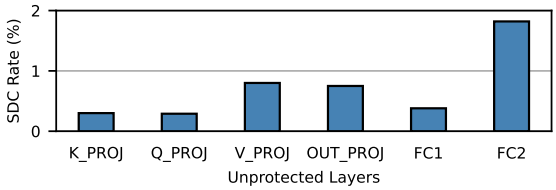


Figure 6: SDC rate after ignoring different layers in protection. Higher bar means the layer is more necessary to protect.

4.1 Identification of Critical Layers

One naïve way to achieve a high SDC coverage for soft errors is to apply the range restriction on all layers. However, due to the large size of the linear layers, protecting every layer may introduce undesirable overhead (nearly 2×). To reduce overhead while achieving high reliability, it is important to select layers that are critical to the overall task accuracy. In this section, we conduct an in-depth characterization study through extensive fault injection experiments to demonstrate the existence of critical layers in LLMs. We then analyze the model architecture to reason about layer criticality and explore a heuristic to automatically identify critical layers without expensive experiments.

To test the criticality of a layer, we apply protection on all linear layers except the layer to be tested, and perform fault injection on all linear layers, to measure whether the LLM is reliable even if we do not protect the tested layer. The fault injection results of applying such protection targeting different layers are shown in Figure 6. A higher SDC rate indicates lower resilience, i.e., worse reliability. Due to the limited space, here we report GPTJ-6B model and SQuAD 2.0 dataset for example. The observations hold for all the models and datasets we consider in this work, and are validated in the Evaluation section (section 5). For layers such as K_PROJ, Q_PROJ, and FC1, the SDC rates vary from 0.29% to 0.38%, indicating that these layers are not critical. There are a considerable amount of SDC outcomes left (0.75% - 1.82%) in V_PROJ, OUT_PROJ, and FC2 layers if we free the protection of these layers. Therefore these are the critical layers.

Table 1 summarizes the identified critical layers (second column). These critical layers are represented by red boxes in Figure 1. We compare different protection selections among existing methods and our FT2 in Table 1. The lack of protection for critical layers may lead to the unsatisfying SDC coverage of existing range-clipping-based methods. With our in-depth characterization, FT2 protects all critical layers and achieves sufficient protection.

Table 1: Layer criticality and the protection coverage of various methods. Shaded lines indicates critical layers. FT2 covers all critical layers. MaxiMals and Global Clipper protect some critical layers, while Ranger does not protect any linear layer.

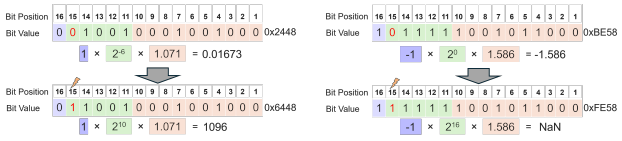
	Critical Layer	Ranger	MaxiMals	Global Clipper	FT2
K_PROJ	N				
Q_PROJ	N				
V_PROJ	Y			✓	✓
OUT_PROJ	Y		✓	✓	✓
FC1	N				
FC2	Y		✓		✓
UP_PROJ	Y				✓
GATE_PROJ	N				
DOWN_PROJ	Y		✓		✓

Take-away #1: Critical layers and non-critical layers co-exist in generative LLMs. Protecting critical layers is necessary to reduce the SDC rate.

4.1.1 *Underlying Reasons for Layer Criticality.*

It is time consuming to perform fault injection experiments to identify critical layers, which is undesired. Here we investigate LLM model architecture and pinpoint two reasons for layer criticality. First, the residual branches affect the criticality of abnormal fault-affected values inside each layer. Secondly, the scaling operation and activation layers affect the distribution of abnormal fault-affected values as well as the layer criticality. We derive insights to enable the possibility of leveraging proxies to identify critical layers.

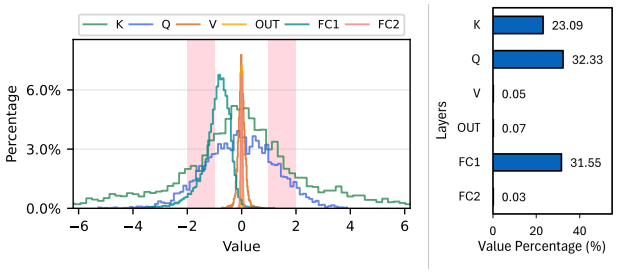
Range restriction can detect two types of abnormal values: out-of-bound values and NaN (Not-a-Number). Below we briefly describe these two abnormal value types using half-precision floating point (FP16) as an example, see Figure 7. An FP16 number has a sign bit (marked in blue), 5 exponent bits (green), and 10 mantissa bits (red). Figure 7(a) shows an example of a bit flip leading to an extremely large value when the highest exponent bit is flipped. An example of NaN is shown in Figure 7(b). In FP16 format, NaN is defined as all exponent bits are 1 and the fraction (represented by mantissa bits) is not 0. Values with specific exponent bits can become NaN after a single-bit flip, and most of them are in the intervals (-2, -1) or (1, 2). We define these two intervals as the *NaN-vulnerable area* and the values fall into these intervals as the *NaN-vulnerable values*.



(a) Extreme value (b) Not A Number (NaN)

Figure 7: Examples of two different types of abnormal values. The bit flip is in the highest exponent bit of FP16 datatype.

Impact of residual branches on the criticality of abnormal values. As shown in Figure 1, residual branches (green-colored arrows) duplicate the inputs of attention block and MLP, then connect to the outputs with an add operation. Correcting NaN to 0 barely corrupts the following computation since the residual connection



(a) Distribution of neuron values among different layers.

(b) Percentage of NaN-vulnerable values.

Figure 8: Neuron value distribution varies among different layers for OPT-6.7B model with SQuAD 2.0 dataset. The percentage of values that fall into NaN-vulnerable Interval is also significantly different.

recovers lost information from the prior part of the model. The correction is also effective when applied to any following critical layer before the residual connection. For example, the NaN in FC1 layer propagates to an entire row in FC2 layer. Although these NaN in FC2 are corrected to 0, the residual fusion can pass important former information forward, ensuring the model can still generate correct output despite the NaN issue. However, the residual connection is unable to mask extreme values, which can propagate to the following layers and finally lead to SDCs.

Take-away #2: Out-of-bound values are more critical than NaN. The effect of NaN is weakened by residual branches and NaN can be corrected in other layers. Extreme out-of-bound values must be corrected right after their originating layer to minimize the propagation.

Our analysis shows that the criticality of out-of-bound values and NaN is different. We profile and draw the neuron value distribution of linear layers (under fault-free execution) to investigate the possibility of faults leading to these two types of abnormal values. Figure 8(a) shows the output value distribution of all 6 linear layers in OPT-6.7B model (block ID 1). The distribution is obtained from a single inference instance using SQuAD 2.0 dataset (input ID 686). We also report the percentage of NaN-vulnerable neuron values in these layers in Figure 8(b). For critical layers including V_PROJ, OUT_PROJ, and FC2, very few neurons fall into NaN-vulnerable intervals, thus non-critical NaNs barely occur when a fault happens in these layers. Neuron values gather tightly between -1 and 1, which may become extreme values if the leading exponent bit is flipped ($0 \rightarrow 1$, as Figure 7(a) shows). For non-critical layers including Q_PROJ, K_PROJ, and FC1, there are a large proportion of neurons that fall into NaN-vulnerable areas (shaded in pink). Recall that NaN is not critical because of residual branches; consisting a large portion of NaN-vulnerable values makes a layer less critical.

Take-away #3: The variety of neuron value distribution leads to different proportions of critical/non-critical abnormal values if faults happen.

Impact of scaling operations and activation layers on layer criticality. We observe that the outputs of non-critical layers K_PROJ and Q_PROJ in the attention block are fed into the attention score calculation which performs scaling down. In other words, the following layer of K_PROJ and Q_PROJ is performing scaling operations where the magnitude of the extreme faulty values are reduced. Therefore, the impact of the error is decreased and the error is more likely to be corrected by the range restriction protections applied in other layers. For the FC1 and GATE_PROJ layers in MLP, the following layer is the activation layer, thus similar magnitude reduction effects happen on extreme faulty values. On the other side, there is no scaling or activation operation in the following layers of critical layers. Thus, extreme out-of-bound values propagate their severe faulty effect to other layers and finally have a higher chance of causing SDCs.

Scaling operations and activation layers affect the neuron value distribution of linear layers. Recall the model architecture shown in Figure 1, non-critical layers are followed by activation layers or scaling operations within attention score calculation, then the next linear layer is always critical. Considering the effect of reducing the value magnitude of scaling operations and activation layers, the wider neuron distribution of non-critical layers is compressed, i.e., to the distribution of critical layer neurons where the majority of values are close to 0. Due to the space constraint, we cannot show the distribution and detailed analysis of every model. We thoroughly examine all the models considered in this work and conclude that our observations hold for all the models.

Take-away #4: The underlying reasons for layer criticality are twofold. First, residual branches reduce NaN criticality; layers with fewer NaN-vulnerable values are more critical. Second, scaling operations and activation layers can reduce the magnitude of faulty values and also affect the distribution of NaN-vulnerable neurons in layers.

4.1.2 Heuristic of Identifying Critical Layers. Inspired by the underlying reasons for layer criticality, we propose two proxies for identifying critical layers: 1) the percentage of NaN-vulnerable values and 2) the existence of scaling operations and activation layers. The percentage of NaN-vulnerable values is calculated by profiling the neuron value distribution, which still requires one undesirable profiling run. On the other side, analyzing the model architecture does not require any heavy execution and can be easily automated. Therefore, FT2 derives layer criticality by the following *heuristic*: a layer is deemed critical if no scaling operation or activation layer is present before the next linear layer. We evaluate and validate this heuristic through 7 LLMs and 3 datasets; see Section 5.2.1 for more details.

Take-away #5: Model architecture contributes to layer criticality. Scaling operations and activation layers can help with limiting the magnitude of faulty values. We derive an efficient heuristic to identify critical layers: a layer is deemed critical if no scaling operation or activation layer is present before the next linear layer.

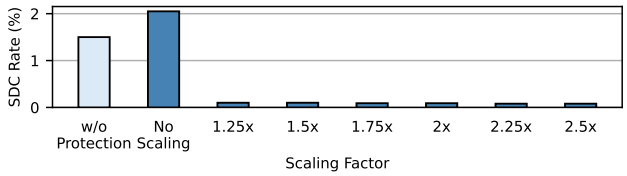


Figure 9: SDC rates using different scaling factors for bounds with Qwen2-7B model and math problem task and GSM8K dataset. After bound scaling, the SDC rate is greatly reduced. This shows that the bound scaling is necessary although FT2 is not sensitive to different scaling factors.

4.2 First-Token Inspired Bound Profiling

Motivated by the bound profiling issues discussed in subsection 3.2, in this section, we explore the possibility of avoiding the expensive offline bound profiling and directly obtaining bounds during the inference. For generative LLMs, the input for generating the first token is a subset of the input for generating the following tokens, therefore the values of neurons may present similarity. This enables our *key idea*: leveraging the information of the first token generation to infer the bounds when generating other tokens.

Leveraging first-token generation for bound obtaining is based on two key insights derived from our study: (1) the bounds we obtain from the generation of the first token with limited data are effective in protecting the execution of generating the following tokens; (2) The effect of soft errors on the first token generation is negligible, therefore the first token generation without range restriction is acceptable.

4.2.1 Effectiveness of bounds profiled from the first token generation.

The two leftmost bars in Figure 9 show the resilience of Qwen2-7B model on GSM8K dataset before and after applying the bound profiled from the first token generation. The SDC rate increases after applying protection. This reflects the challenge of limited online data: the data from generating the first token is too few to form effective and stable bounds. The bounds are not large enough to capture normal values. Protection using these limited bounds may wrongly clip normal values, leading to more incorrect outputs.

We scale the bounds to overcome the challenge of limited online data, inspired by the bound scaling initially used in MaxiMals [57]. The SDC rate of injecting faults into Qwen2-7B model after applying various scaling factors is also shown in Figure 9. We observe that the SDC rate dramatically reduces after bound scaling. Even with a slight scaling of 1.25x, the error resilience is greatly improved. We conclude that performing bound scaling is necessary, but FT2 is not sensitive to the choice of scaling factor. In FT2 the scaling factor is set to 2 for easy and faster calculation.

Take-away #6: The bounds profiled from the first token generation (with bound scaling) are effective in protecting the generation of the following tokens. Our intuition for bound scaling is to overcome the limited online data where insufficient bounds are profiled and normal values could be clipped.

4.2.2 Impact of soft errors on the first token generation.

In this section, we demonstrate that the impact of soft errors on the first token generation is negligible. Firstly, the probability of soft errors occurring during the first token generation is low. Meanwhile, even if an error occurs during the first token generation, an LLM can still achieve high resilience without bounds, given that we can still detect and correct NaN. Last but not least, fault-affected bounds are still effective. We expand our argument below.

Firstly, we argue that the probability of soft errors occurring during the first token generation is low. Considering the origin of soft errors such as radioactive particles generated from cosmic rays and device aging, the probability of soft errors happening at a certain time is the same. Specifically, if a soft error occurs in one LLM inference execution, the probability of soft errors occurring in its first token generation phase is the proportion of the execution time of the first token generation. Figure 10 shows several examples of the execution time percentage of the first token generation over the LLM inference, with different datasets and models on two hardware configurations. Consistent with all the experiments in this work, we generate 60 tokens for QA tasks and 180 tokens for Math task in total (i.e., 1 first token plus 59/179 following tokens). The first token generation takes less than 9% of the total inference time. In detail, the percentage of the execution time for the first token generation on an NVIDIA A100 GPU is 1.89% - 8.33% for QA tasks (SQuAD 2.0 and XTREME datasets), while for Math task (GSM8K dataset) the percentage is 0.6% - 2.66%. When using an NVIDIA H100 GPU, the percentages of execution time are 1.75% - 2% and 0.59% - 0.61%, respectively. The first token generation takes slightly longer time than the average token generation time because the input is larger in the first iteration.

Secondly, the resilience of the first token generation is already satisfying with NaN correction. Although it is not possible to detect and correct extreme abnormal values due to the unavailability of bounds, NaN can always be detected and corrected. We back up this insight with resilience results in Figure 11 with SQuAD 2.0 dataset and OPT-6.7B model. We conduct a fault injection campaign by injecting faults during the first token generation only to assess the impact of soft errors on the first token generation. The leftmost bar of each group shows the LLM resilience without protection where faults can be injected into the generation of any tokens. Compared to the resilience of applying the complete protection with FT2 (the middle bar of each group), the resilience of the first token generation is already at the same level with NaN corrected. Even for the most challenging fault model EXP where the exponent bits are flipped, the resilience of the first token generation is still satisfying. The model can generate correct output even if the first token is different, because the first token is often a meaningless special token, e.g., “the”.

Lastly, fault-affected bounds are still effective when a fault occurs at the generation of the first token (assuming that NaN is corrected), only the layer that encounters the occurrence of the fault would experience a significant change in its bounds. The bounds of other layers may change slightly, therefore are still effective in detecting and correcting abnormal values. Considering the rareness of two hardware faults happening at one LLM inference and on the execution of the same layer, the probability of the second fault occurs on the same fault-affected layer with the faulty bound is negligible.

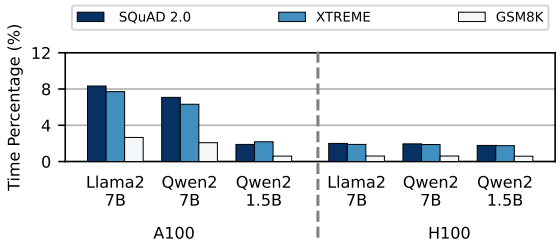


Figure 10: The percentage of execution time for the first token generation. Generating the first token takes less than 10% of the total execution time.

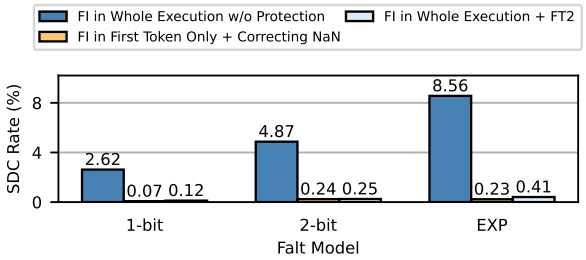


Figure 11: The resilience of the first token generation is high. Correcting NaN in the first token generation can further reduce SDC rate to the same level as applying full protection to the following generation, which is negligible.

Take-away #7: The bounds profiled from the first token generation are still effective even with the presence of a soft error at first token generation.

4.3 Applying Protection

After finishing the generation of the first token, FT2 assumes to have the bounds for the rest of the tokens. Protection is applied after the execution of each critical layer. FT2 checks all the neuron values within each critical layer to detect abnormal values.

Protection through range restriction only detects/corrects large abnormal values. This is because most neurons are not large in magnitude and small faulty values barely lead to SDCs. However, large faulty values easily trigger SDC outcomes. Figure 7(b) shows an example of a bit flip leading to an extremely large value when the highest exponent bit is flipped. We clip the extreme out-of-bound values to the bound to conservatively reflect the large (but normal) values in generative LLM tasks. For example, Figure 12 shows the value distribution of neurons in DOWN_PROJ, UP_PROJ, and GATE_PROJ layers of Vicuna-7B model (block ID 1) with SQuAD 2.0 dataset (input ID 686). Most values are close to 0 for both layers, but there are a few large values in DOWN_PROJ layer. Note that this value distribution is profiled from a fault-free execution, and these outliers may exceed the bounds after fault injection. Therefore, clipping these values to 0 would result in significant deviation, which may lead to an incorrect output. Using distribution-based sampling methods [42] leads to similar huge deviations and wrong outputs, since most neuron values are close to 0. In contrast, clipping the outliers to bounds is more appropriate in handling these large values

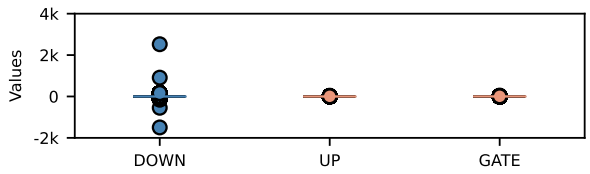


Figure 12: The neuron value distributions of three layers from Vicuna-7B model with QA task and SQuAD 2.0 dataset. Large neuron values exist in some layers of generative LLMs.

and can maintain model performance. We emphasize that the existence of large neuron values is a unique characteristic of generative LLM tasks. Existing range-restriction-based solutions focus on image tasks where extreme neuron values are very few [12, 42, 57, 60], leading to small bound values which are close to 0. Clipping those out-of-bound values to 0, bounds, or sample values has a similar effect since the deviation is negligible in these image tasks.

Take-away #8: Due to the existence of large neuron values in generative LLMs, extreme out-of-bound values must be clipped to the bound.

Implementation. Our protection method is implemented by only using the default PyTorch functions. We employ *torch.clamp* and *torch.nan_to_num* to correct the out-of-bound values and NaN values, respectively. To lower the runtime overhead, we fused these two operations into a new Torch-level function based on *torch.clamp* by modifying less than 20 lines of C++ code. Then we implement the dispatch registration to make the functions directly callable at the PyTorch level. Our protection function can be applied to any model by simply calling it after the identified critical layers.

5 EVALUATION

In this section, we evaluate the protection efficiency of FT2 and compare it with several baselines. We also measure the inference time and memory overhead of FT2. Besides, we discuss the sensitivity of protection efficiency under different datatype and hardware configurations. We describe our experimental set-up, followed by the evaluation results.

5.1 Experimental Set-up

All the experiments are performed on two hardware configurations. The first consists of AMD EPYC 7742 64-Core CPU and NVIDIA A100 GPU (Ampere architecture), running Rocky Linux 8.10 as its operating system. The second is NVIDIA GH200 Grace Hopper Superchip, consisting of NVIDIA Grace 72 Arm Neoverse V2 cores CPU and NVIDIA H100 GPU (Hopper architecture) on Rocky Linux 9.3. Unless stated otherwise, results are based on A100 GPUs. The reliability results are independent of hardware, as we evaluate in Section 5.2.4.

We select generative tasks with definite correct answers including question-answering (QA) and mathematical reasoning so that the decision of fault injection experiment outcome (i.e., Masked or SDC) can be automated. For generative tasks such as translation and summarization, automating the semantic correctness checking is difficult since there exist multiple correct ways of answering the same question. We consider FP16 models as FP16 is faster than FP32 but has the same level of model quality. Table 2 shows the models and tasks considered in this work. GPTJ-6B [67], OPT-6.7B [78], and OPT-2.7B [78] models follow the model architecture shown in Figure 1(a); the architecture of Llama2-7B [66], Vicuna-7B [14], Qwen2-7B [70], and Qwen2-1.5B [70] models is the same as Figure 1(b). We consider SQuAD 2.0 [54] and Google XTREME [28] for question-answering tasks. For math problem solving, we use the GSM8K math problem solving dataset [15]. Note that among these models, only Llama2-7B model and Qwen2-7B model perform relatively well at answering math problems, thus only these two models are considered for math problems. Due to the limited computation resources and the substantial resource requirements of tested generative LLMs, we randomly choose 50 inputs from the set of inputs where all models produce correct outputs for these questions. We performed 500 fault injection campaigns per input (approximately ±0.00554% - ±0.368% error margins across different models with a 95% confidence interval [32, 33]), amounting to over 11 million fault injections (around 8000 GPU hours) for all characterization and evaluation results.

To determine the token generation length, we evaluate the last position of the correct answer for all fault-free outputs, which is 50 and 150 for QA and math, respectively. We extend the output length to 120% to cover cases where fault-injected inferences generate correct but longer answers. This results in generating 60 tokens for QA tasks and 180 tokens for the Math task.

Baselines. We compare our method with several existing range-restriction-based protection methods, including Ranger [12], MaxiMals [57], and Global Clipper [60]. Ranger is implemented on TensorFlow; the codes of MaxiMals and Global Clipper are not publicly available and no implementation details are given. We implement these three baselines on PyTorch to replicate their protection functionality. Due to the lack of implementation details, we cannot replicate their runtime performance. Therefore, we can only compare the resilience (i.e., protection effectiveness) of these baselines, but not overhead. We also use FT2 with bounds obtained from the expensive offline profiling to evaluate the effectiveness of our first-token-inspired bounds. The bounds are obtained from 20% of the training dataset, which is in line with prior works [12, 60].

5.2 Evaluation Results

We present our evaluation results starting from the overall effectiveness and overhead. Then, we discuss the sensitivity of FT2 from various aspects, including datasets, data types, and hardware.

5.2.1 *Effectiveness: Overall SDC Rate.*

Figure 13 shows the resilience of applying FT2 , compared with all the baselines. A lower bar indicates the model is more reliable,

Table 2: Models and tasks for evaluation.

Model Name	# of Parameters	Task Type
OPT-6.7B [78]	6.66B	QA
OPT-2.7B [78]	2.65B	QA
GPTJ-6B [67]	6.05B	QA
Llama2-7B [66]	6.74B	QA/Math
Vicuna-7B (v1.5) [14]	6.74B	QA
Qwen2-7B [70]	7.62B	QA/Math
Qwen2-1.5B [70]	1.54B	QA

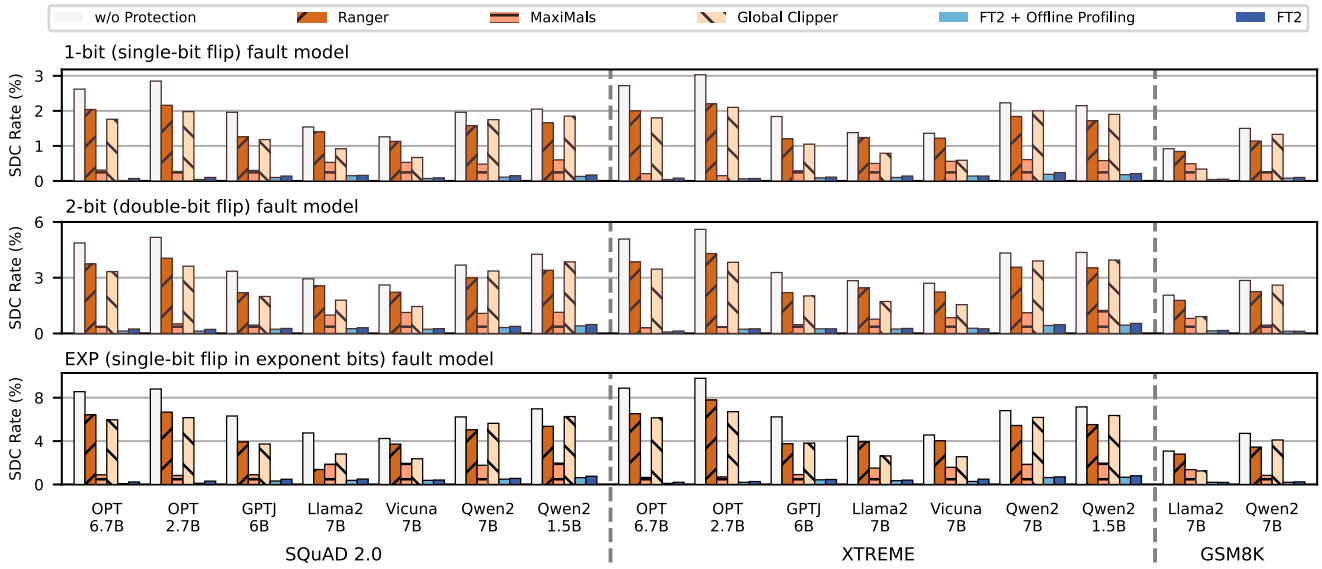


Figure 13: SDC rate comparison of FT2 against baselines. FT2 is effective in protecting generative LLM inference.

i.e., the protection is more effective. Each row represents a different fault model, with results grouped by datasets within a row. Our FT2 outperforms other baselines significantly in protection effectiveness, achieving an average of 92.92% SDC rate reduction.

Among all baselines, Ranger performs the worst with an average SDC rate of 2.83% since it only protects the activation layers, ignoring the critical linear layers. Global Clipper also suffers from an average SDC rate of 2.61% because the critical linear layers in MLP are not protected by it. MaxiMals decreases the SDC rate significantly to an average of 0.81%, but it is not effective on Llama2-7B, Qwen2-7B, and Qwen2-1.5B because MaxiMals does not protect the critical UP_PROJ layers in these models. FT2 with and without offline bound profiling achieve comparable SDC rates of 0.204% and 0.25% on average across all benchmarks, indicating the effectiveness of the first-token-inspired approach employed by FT2 .

Overall resilience trends and observations hold under all three fault models considered in this study. In particular, LLM inference with 1-bit error exhibits the lowest SDC rates; the EXP model is the most aggressive one with the highest SDC rates. After applying FT2 , the SDC rate is significantly decreased considering all 3 fault models. Even for the worst case of OPT-2.7B model with XTREME dataset under EXP fault model, applying FT2 can reduce the SDC rate from 9.79% to 0.27%, which is a significant improvement in fault tolerance. Therefore, FT2 is demonstrated to successfully reduce the SDC outcomes of generative LLM inference against different fault types.

Considering different datasets, the average SDC rate on SQuAD 2.0 dataset after applying FT2 is 0.3%, which is almost the same with XTREME dateset (0.31%). The SDC rate drops to 0.14% on average for GSM8K dataset. Thus, FT2 can protect LLM inference of different datasets and generative tasks effectively.

5.2.2 Efficiency: Execution Overhead.

Figure 14 shows the execution time overhead of applying our protections. Experiments are performed on an A100 GPU and each

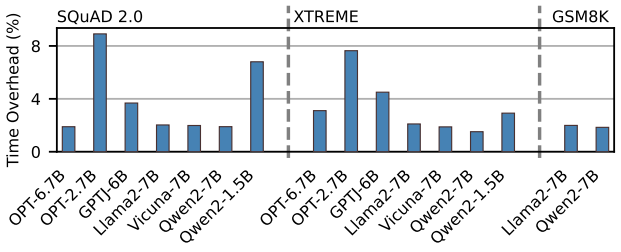


Figure 14: Time overhead of FT2 on an NVIDIA A100 GPU.

experiment is repeated 1,000 times. There is no observable time difference among those repeated experiments. On average, FT2 introduces 3.42% runtime overhead, which is lower than the 5.61% runtime overhead of MaxiMals reported in their original paper [57]. Note that no overhead number is reported by Global Clipper. Even for the worst case of OPT-2.7B models, the overhead is still less than 9% (8.91%). The execution time of each inference instance varies from 1.35 to 6.4 seconds, while protection takes an addition of 32.5 to 127.5 milliseconds.

Memory overhead is negligible (288 - 512 Bytes, <0.2% for all models) since only two bound values are stored for each layer. Each model has around 72 - 128 protected layers counting all the layers in different blocks.

5.2.3 Sensitivity to Data Types. We evaluate the effectiveness of our protection on different data types, FP16 and FP32, as a case study. Figure 15 shows the SDC rate comparison of protection techniques when using FP16 and FP32, for OPT-6.7B and GPTJ-6B models with SQuAD 2.0 dataset. After applying FT2, the SDC rate drops to 0.14%, showing that FT2 is also effective in protecting FP32 LLM inference.

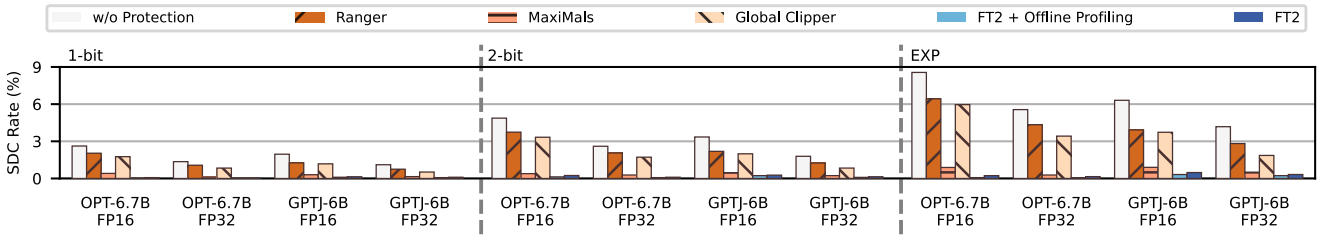


Figure 15: SDC rate comparison of applying FT2 against baselines considering two data types, FP16 and FP32.

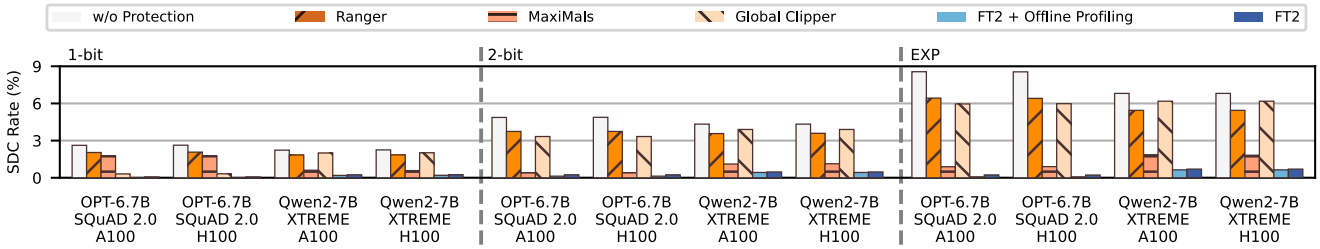


Figure 16: SDC rate of applying FT2 against baselines for two hardwares, NVIDIA A100 and H100 GPUs.

5.2.4 Sensitivity to Hardware. As a case study, we evaluate the protection effectiveness of FT2 on two different hardware configurations with NVIDIA A100 and H100 GPU respectively. Here we evaluate the OPT-6.7B model on SQuAD 2.0 dataset and the Qwen2-7B model on XTREME dataset, and the SDC rates are shown in Figure 16. Overall, the SDC rate of LLM inference on H100 is the same as that on A100. After applying FT2, the SDC rate drops to 0.33% for H100, which is the same as A100, indicating the feasibility of our method among different generations of NVIDIA GPUs. Since FT2 is a software-level solution, we do not foresee any particular issues of applying it to other hardware settings, such as CPUs and AMD GPUs.

6 RELATED WORK

There are plenty of studies focusing on measuring the reliability of neural networks [2, 17, 19, 34, 35, 43, 50, 57], delivering insights for the mitigation of soft errors. For transformers, Ma et al. combine fault injection and algorithmic checksum to assess the resilience of each component in transformer-based models [43]. Agarwal et al. perform register-transfer-level fault injection to analyze the reliability of LLMs on multiple tasks [2]. Roquest et al. perform a beam test to measure fault effects on vision transformers [57]. Different from the above studies, our characterization study focuses on criticality assessment and provides insights that guide the design of protection methodologies.

Reliability protection of neural networks can be deployed at different levels. Hardware protections such as Error Correction Code (ECC) [25, 27, 61] are widely used in modern GPUs that perform single error correction and double error detection (SEC-DED). Standard ECC cannot correct multi-bit errors and cannot cover computation errors in logical units such as ALUs. At the software level, Triple Modular Redundancy (TMR) is a standard technique to improve resilience [47, 71]. Schmedding et al. selectively protect weights in CNN models based on an important score [58]. Instead of simply duplicating weights, Structural Coding adds linear combinations of parameter groups to detect and mitigate errors with low overhead [4]. Algorithm-based Fault Tolerance (ABFT) methods are used to protect transformers by encoding a checksum into the computation process [37, 38, 40]. The above methods provide high reliability but with high overhead.

Several range-restriction-based protection methods are developed for neural networks [12, 57, 60, 76]. Ranger [12] protects only the output of activation layers. Zhan et al. also restrict the value of activation layers, but the protection target is to mitigate soft errors in memory [76]. These two solutions are not fully adequate for generative LLMs because critical linear layers exist. This is also confirmed with the high SDC rate applying Ranger (see subsubsection 5.2.1). Designed for LLMs, Global Clipper [60] protects only the output of linear layers in the attention block, wrongly ignoring the critical layers in MLPs. MaxiMals [57] protects the output of attention blocks and MLPs but ignores critical layers such as V_PROJ and UP_PROJ. Besides the deficiency of protection, all existing range-restriction-based methods require costly profiling on the training dataset to acquire bounds.

In contrast, FT2 identifies and protects all critical layers based on our thorough characterization study. FT2 also performs online-only protection instead of expensive offline profiling, leveraging the bounds obtained during the first token generation. *To our best knowledge, FT2 is the first complete protection for generative LLMs without offline profiling while achieving high reliability.*

7 CONCLUSION

We demonstrate that existing resilience enhancement solutions have fundamental limitations, for example, high overhead. State-of-the-art range-restriction-based solutions offer negligible protection overhead but have issues such as incomplete protection coverage

and mandatory offline profiling as we point out. To address this, we design FT2, a First-Token-inspired online Fault Tolerance methodology on critical layers for generative LLMs, offering high reliability and low overhead. We thoroughly analyze the layer criticality based on our extensive fault injection campaign with over 11 million experiments and identify critical layers: a layer is deemed critical if no scaling operation or activation layer is present before the next linear layer. To achieve online protection, FT2 leverages a key insight: the input of generating the first token is a subset of the input of generating the following tokens for generative LLM inference. Therefore, in FT2, bounds are profiled during the first token generation and then used to protect the generation of the following tokens. We extensively evaluate FT2 across 7 LLMs and 3 datasets under 3 fault models. FT2 outperforms the existing protection mechanisms and achieves 92.92% SDC rate reduction with only 3.42% overhead on average.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation (NSF) grants (#2402940 and #2410856) and the Commonwealth Cyber Initiative (CCI) grant (#HC-3Q24-047). The platforms used for evaluation in this work are supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, under award 66150: "CENATE - Center for Advanced Architecture Evaluation". The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract DE-AC05-76RL01830. This project is supported by resources provided by the Office of Research Computing at George Mason University (URL: https://orc.gmu.edu) and funded in part by grants from the National Science Foundation (Award Number 2018631). The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing computational resources that have contributed to the research results reported within this paper. URL: http://www.tacc.utexas.edu

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Udit Kumar Agarwal, Abraham Chan, and Karthik Pattabiraman. 2023. Resilience Assessment of Large Language Models under Transient Hardware Faults. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 659–670.

[3] Fatih Kadir Akin. 2022. Awesome ChatGPT Prompts. https://huggingface.co/datasets/fka/awesome-chatgpt-prompts.

[4] Ali Asgari Khoshouyeh, Florian Geissler, Syed Qutub, Michael Paulitsch, Prashant Nair, and Karthik Pattabiraman. 2023. Structural coding: A low-cost scheme to protect cnns from large-granularity memory faults. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–17.

[5] Rizwan A Ashraf, Roberto Gioiosa, Gokcen Kestor, Ronald F DeMara, Chen-Yong Cher, and Pradip Bose. 2015. Understanding the propagation of transient errors in HPC applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[6] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732* (2021).

[7] Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa-Anke, and Leonardo Neves. 2020. TweetEval:Unified Benchmark and Comparative Evaluation for Tweet Classification. In *Proceedings of Findings of EMNLP*.

[8] Robert Baumann. 2005. Soft errors in advanced computer systems. *IEEE design & test of computers* 22, 3 (2005), 258–266.

[9] Jakub Breier, Dirmanto Jap, Xiaolu Hou, Shivam Bhasin, and Yang Liu. 2021. SNIFF: reverse engineering of neural networks with fault attacks. *IEEE Transactions on Reliability* 71, 4 (2021), 1527–1539.

[10] Nandhini Chandramoorthy, Karthik Swaminathan, Martin Cochet, Arun Paidimarri, Schuyler Eldridge, Rajiv V Joshi, Matthew M Ziegler, Alper Buyuktosunoglu, and Pradip Bose. 2019. Resilient low voltage accelerators for high energy efficiency. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 147–158.

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[12] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. 2021. A low-cost fault corrector for deep neural networks through range restriction. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 1–13.

[13] Zitao Chen, Niranjhana Narayanan, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2020. Tensorfi: A flexible fault injection framework for tensorflow applications. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 426–435.

[14] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)* 2, 3 (2023), 6.

[15] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021).

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[17] Fernando Fernandes dos Santos, Caio Lunardi, Daniel Oliveira, Fabiano Libano, and Paolo Rech. 2019. Reliability evaluation of mixed-precision architectures. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 238–249.

[18] Bo Fang, Xinyi Li, Harvey Dam, Cheng Tan, Siva Kumar Sastry Hari, Timothy Tsai, Ignacio Laguna, Dingwen Tao, Ganesh Gopalakrishnan, Prashant Nair, et al. 2024. Understanding Mixed Precision GEMM with MPGemmFI: Insights into Fault Resilience. In *2024 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 166–178.

[19] Bo Fang, Xinyi Li, Harvey Dam, Cheng Tan, Siva Kumar Sastry Hari, Timothy Tsai, Ignacio Laguna, Dingwen Tao, Ganesh Gopalakrishnan, Prashant Nair, Kevin Barker, and Ang Li. 2024. Understanding Mixed Precision GEMM with MPGemmFI: Insights into Fault Resilience. In *2024 IEEE International Conference on Cluster Computing (CLUSTER)*. 166–178. https://doi.org/10.1109/CLUSTER59578.2024.00022

[20] Bo Fang, Karthik Pattabiraman, Matei Ripeanu, and Sudhanva Gurumurthi. 2014. GPU-Qin: A methodology for evaluating the error resilience of GPGPU applications. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 221–230.

[21] Vinicius Fratin, Daniel A. G. de Oliveira, Caio B. Lunardi, Fernando Santos, Gennaro Rodrigues, and Paolo Rech. 2018. Code-Dependent and Architecture-Dependent Reliability Behaviors. In *DSN*. 13–26.

[22] Giorgis Georgakoudis, Ignacio Laguna, Dimitrios S Nikolopoulos, and Martin Schulz. 2017. Refine: Realistic fault injection via compiler-based instrumentation for accuracy, portability and speed. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[23] Hui Guan, Lin Ning, Zhen Lin, Xipeng Shen, Huiyang Zhou, and Seung-Hwan Lim. 2019. In-place zero-space memory protection for cnn. *Advances in Neural Information Processing Systems* 32 (2019).

[24] Perttu Hämäläinen, Mikke Tavast, and Anton Kunnari. 2023. Evaluating large language models in generating synthetic hci research data: a case study. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–19.

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

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

[27] Mu-Yue Hsiao. 1970. A class of optimal minimum odd-weight-column SEC-DED codes. *IBM Journal of Research and Development* 14, 4 (1970), 395–401.

[28] Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. XTREME: A Massively Multilingual Multi-task Benchmark for Evaluating Cross-lingual Generalization. *CoRR* abs/2003.11080 (2020). arXiv:2003.11080

[29] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou

Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[30] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[31] Joyjit Kundu, Wenzhe Guo, Ali BanaGozar, Udari De Alwis, Sourav Sengupta, Puneet Gupta, and Arindam Mallik. 2024. Performance modeling and workload analysis of distributed large language model training and inference. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 57–67.

[32] Lawrence M Leemis and Stephen Keith Park. 2006. *Discrete-event simulation: A first course*. Pearson Prentice Hall Upper Saddle River, NJ. pg. 366.

[33] Régis Leveugle, A Calvez, Paolo Maistri, and Pierre Vanhauwaert. 2009. Statistical fault injection: Quantified error and confidence. In *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 502–506.

[34] Jonathan Lew, Deval A Shah, Suchita Pati, Shaylin Cattell, Mengchi Zhang, Amruth Sandhupatla, Christopher Ng, Negar Goli, Matthew D Sinclair, Timothy G Rogers, et al. 2019. Analyzing machine learning workloads using a detailed GPU simulator. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 151–152.

[35] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W Keckler. 2017. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[36] Guanpeng Li and Karthik Pattabiraman. 2018. Modeling input-dependent error propagation in programs. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 279–290.

[37] Yuhang Liang, Xinyi Li, Jie Ren, Ang Li, Bo Fang, and Jieyang Chen. 2024. Light-Weight Fault Tolerant Attention for Large Language Model Training. *arXiv preprint arXiv:2410.11720* (2024).

[38] Yuhang Liang, Xinyi Li, Jie Ren, Ang Li, Bo Fang, and Jieyang Chen. 2025. ATTNChecker: Highly-Optimized Fault Tolerant Attention for Large Language Model Training. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (Las Vegas, NV, USA) *(PPoPP '25)*. Association for Computing Machinery, New York, NY, USA, 252–266. https://doi.org/10.1145/3710848.3710870

[39] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 740–755.

[40] Haoxuan Liu, Vasu Singh, Michał Filipiuk, and Siva Kumar Sastry Hari. 2024. ALBERTA: ALgorithm-Based Error Resilience in Transformer Architectures. *IEEE Open Journal of the Computer Society* (2024).

[41] Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. 2024. On llms-driven synthetic data generation, curation, and evaluation: A survey. *arXiv preprint arXiv:2406.15126* (2024).

[42] Dongning Ma, Fred Lin, Alban Desmaison, Joel Coburn, Daniel Moore, Sriram Sankar, and Xun Jiao. 2024. Dr. DNA: Combating Silent Data Corruptions in Deep Learning using Distribution of Neuron Activations. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 239–252.

[43] Kwondo Ma, Chandramouli Amarnath, and Abhijit Chatterjee. 2023. Error Resilient Transformers: A Novel Soft Error Vulnerability Guided Approach to Error Checking and Suppression. In *2023 IEEE European Test Symposium (ETS)*. IEEE, 1–6.

[44] Abdulrahman Mahmoud, Neeraj Aggarwal, Alex Nobbe, Jose Vicarte, Sarita Adve, Christopher Fletcher, Iuri Frosio, and Siva Hari. 2020. PyTorchFI: A Runtime Perturbation Tool for DNNs. 25–31. https://doi.org/10.1109/DSN-W50199.2020.00014

[45] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari. 2020. PyTorchFI: A Runtime Perturbation Tool for DNNs. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 25–31.

[46] Abdulrahman Mahmoud, Neeraj Aggarwal, Alex Nobbe, Jose Rodrigo Sanchez Vicarte, Sarita V Adve, Christopher W Fletcher, Iuri Frosio, and Siva Kumar Sastry Hari. 2020. Pytorchfi: A runtime perturbation tool for dnns. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 25–31.

[47] Andrew Milluzzi and Alan George. 2017. Exploration of TMR fault masking with persistent threads on Tegra GPU SoCs. In *2017 IEEE Aerospace Conference*. IEEE, 1–7.

[48] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196* (2024).

[49] Saif Mohammad, Felipe Bravo-Marquez, Mohammad Salameh, and Svetlana Kiritchenko. 2018. Semeval-2018 task 1: Affect in tweets. In *Proceedings of the 12th international workshop on semantic evaluation*. 1–17.

[50] Niranjhana Narayanan, Zitao Chen, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2023. Fault Injection for TensorFlow Applications. *IEEE Transactions on Dependable and Secure Computing* 20, 4 (2023), 2677–2695. https://doi.org/10.1109/TDSC.2022.3175930

[51] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H Rogers. 2016. A large-scale study of soft-errors on GPUs in the field. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 519–530.

[52] Bin Nie, Ji Xue, Saurabh Gupta, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. 2017. Characterizing Temperature, Power, and Soft-Error Behaviors in Data Center Systems: Insights, Challenges, and Opportunities. In *MASCOTS 2017*. 22–31.

[53] Vladyslav Oles, Anna Schmedding, George Ostrouchov, Woong Shin, Evgenia Smirni, and Christian Engelmann. 2024. Understanding GPU Memory Corruption at Extreme Scale: The Summit Case Study. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 188–200.

[54] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, Melbourne, Australia, 784–789. https://doi.org/10.18653/v1/P18-2124 arXiv:1806.03822 [cs.CL]

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

[56] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.

[57] Lucas Roquet, Fernando Fernandes dos Santos, Paolo Rech, Marcello Traiola, Olivier Sentieys, and Angeliki Kritikakou. 2024. Cross-Layer Reliability Evaluation and Efficient Hardening of Large Vision Transformers Models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[58] Anna Schmedding, Lishan Yang, Adwait Jog, and Evgenia Smirni. 2024. Aspis: Lightweight Neural Network Protection Against Soft Errors. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 248–259.

[59] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Efficient on-line error detection and mitigation for deep neural network accelerators. In *Computer Safety, Reliability, and Security: 37th International Conference, SAFECOMP 2018, Västerås, Sweden, September 19-21, 2018, Proceedings 37*. Springer, 205–219.

[60] Qutub Syed Sha, Michael Paulitsch, Karthik Pattabiraman, Korbinian Hagn, Fabian Oboril, Cornelius Buerkle, Kay-Ulrich Scholl, Gereon Hinz, and Alois Knoll. 2024. Global Clipper: Enhancing Safety and Reliability of Transformer-based Object Detection Models. *arXiv preprint arXiv:2406.03229* (2024).

[61] Charles W Slayman. 2005. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *IEEE Transactions on Device and Materials Reliability* 5, 3 (2005), 397–404.

[62] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990* (2022).

[63] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[64] Jörg Tiedemann. 2012. Parallel Data, Tools and Interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association (ELRA), Istanbul, Turkey, 2214–2218. http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf

[65] Devesh Tiwari, Saurabh Gupta, George Gallarno, Jim Rogers, and Don Maxwell. 2015. Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*. 38:1–38:12. https://doi.org/10.1145/2807591.2807666

[66] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

[67] Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 billion parameter autoregressive language model.

[68] Jiesheng Wei, Anna Thomas, Guanpeng Li, and Karthik Pattabiraman. 2014. Quantifying the accuracy of high-level fault injection techniques for hardware faults. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 375–382.

[69] Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. 2024. DeepSeek-Prover: Advancing Theorem Proving in LLMs through Large-Scale Synthetic Data. *arXiv preprint arXiv:2405.14333* (2024).

[70] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 Technical Report. arXiv:2407.10671 [cs.CL] https://arxiv.org/abs/2407.10671

[71] Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. 2021. Enabling Software Resilience in GPGPU Applications via Partial Thread Protection. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1248–1259.

[72] Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. 2021. Practical Resilience Analysis of GPGPU Applications in the Presence of Single- and Multi-Bit Faults. *IEEE Trans. Comput.* 70, 1 (2021), 30–44. https://doi.org/10.1109/TC.2020.2980541

[73] Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. 2021. Practical Resilience Analysis of GPGPU Applications in the Presence of Single- and Multi-Bit Faults. *IEEE Trans. Comput.* 70, 1 (2021), 30–44.

[74] Lishan Yang, Bin Nie, Adwait Jog, and Evgenia Smirni. 2021. SUGAR: Speeding Up GPGPU Application Resilience Estimation with Input Sizing. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 1 (2021), 1–29.

[75] Lishan Yang, George Papadimitriou, Dimitris Sartzetakis, Adwait Jog, Evgenia Smirni, and Dimitris Gizopoulos. 2024. GPU Reliability Assessment: Insights Across the Abstraction Layers. In *2024 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 1–13.

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

[77] Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. 2020. Improving Massively Multilingual Neural Machine Translation and Zero-Shot Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 1628–1639. https://doi.org/10.18653/v1/2020.acl-main.148

[78] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068 [cs.CL] https://arxiv.org/abs/2205.01068