FKeras: A Sensitivity Analysis Tool for Edge Neural Networks

OLIVIA WENG and ANDRES MEZA, University of California San Diego, USA
QUINLAN BOCK and BENJAMIN HAWKS, Fermi National Accelerator Laboratory, USA
JAVIER CAMPOS and NHAN TRAN, Fermi National Accelerator Laboratory, USA
JAVIER MAURICIO DUARTE and RYAN KASTNER, University of California San Diego, USA

Edge computation often requires robustness to faults, e.g., to reduce the effects of transient errors and to function correctly in high radiation environments. In these cases, the edge device must be designed with fault tolerance as a primary objective. `FKeras` is a tool that helps design fault-tolerant edge neural networks that run entirely on chip to meet strict latency and resource requirements. `FKeras` provides metrics that give a bit-level ranking of neural network weights with respect to their sensitivity to faults. `FKeras` includes these sensitivity metrics to guide efficient fault injection campaigns to help evaluate the robustness of a neural network architecture. We show how to use `FKeras` in the co-design of edge NNs trained on the high-granularity endcap calorimeter dataset, which represents high energy physics data, as well as the CIFAR-10 dataset. We use `FKeras` to analyze a NN's fault tolerance to consider alongside its accuracy, performance, and resource consumption. The results show that the different NN architectures have vastly differing resilience to faults. `FKeras` can also determine how to protect neural network weights best, e.g., by selectively using triple modular redundancy on only the most sensitive weights, which reduces area without affecting accuracy.

CCS Concepts: • **Hardware → Transient errors and upsets**; *Application specific integrated circuits*; • **Computing methodologies → Neural networks**.

Additional Key Words and Phrases: Fault-tolerant neural networks, edge AI, hardware-software codesign

ACM Reference Format:
Olivia Weng, Andres Meza, Quinlan Bock, Benjamin Hawks, Javier Campos, Nhan Tran, Javier Mauricio Duarte, and Ryan Kastner. 2024. `FKeras`: A Sensitivity Analysis Tool for Edge Neural Networks . In . ACM, New York, NY, USA, 26 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Machine learning (ML) is increasingly used in safety-critical applications, including autonomous vehicles [8, 12, 66], healthcare [1, 3, 63], and scientific experiments [14, 23, 31]. In these domains, the ML computation must act reliably in the face of errors. Soft errors are a common source of unreliability [9, 25, 37, 38], which are difficult to avoid and often require mitigation. For example, a particle strike can cause a bit flip in neural network (NN) weights, which can lead to incorrect results.

Prior work has investigated the fault tolerance of NNs extensively; however, most studies have focused on faults occurring in the outputs of NN layers, namely in pipeline registers and arithmetic logic units (ALUs). They also assume that the weights reside in memory that is protected by error correction codes or parity [4, 18, 46]. This is often the case for large NNs that rely on retrieving weights from off-chip memory, e.g., DRAM, when they are run on GPUs because all the weights cannot fit on chip [51]. This "off-chip inference" is a popular case, especially as researchers add increasingly more parameters to improve NN performance, e.g., GPT-3 has 175 billion parameters, which amounts to 350 GB of data when stored with float16 precision [11, 73]. By focusing on NNs that execute via off-chip inference, previous studies have neglected to study fault tolerance in an emerging class of NNs where inference occurs *completely on chip* [6, 10].

On-chip inference, wherein a NN executes entirely on chip, is increasing in demand, particularly in scientific fields like high energy physics and electron microscopy [10, 23, 24, 31, 72]. As scientific instruments improve, they are able to produce data at extremely high rates, e.g., TBs/s. With so much data, scientists are contending with how to best collect and analyze so much data. Many are looking to NNs to process the data because traditional methods have reached their limits [73].

Consider the CERN Large Hadron Collider (LHC) Compact Muon Solenoid (CMS) experiment [15], which runs particle collision experiments that generate data rates of ~40 TB/s. To reduce data rates, LHC physicists plan to deploy tens of thousands of encap concentrator (ECON-T) ASICs [24], each running a NN encoder, to compress experimental data from the high-granularity endcap calorimeter (HGCAL) [21] into a smaller format for easy filtering in the trigger system. The ECON-T encoder hardware must accept new input data at 40 MHz and complete inference in 25 ns within an area budget of 4 mm^2 [24]. To meet these constraints, the ECON-T:

(1) is a small two-layer NN with ~2000 parameters,
(2) is quantized to have 6-bit fixed-point weights, and
(3) operates completely on-chip.

To complicate matters further, the ECON-Ts operate in a high-radiation environment due to their close proximity to particle collisions in the LHC. High radiation causes transient hardware errors, which can lead to incorrect application output (silent data corruptions) if the hardware is not designed robustly. The ECON-Ts filter terabytes per second of data for high-energy physics studies, and faulty execution is unacceptable. Only the NN weight parameters are vulnerable to faults because the activations are not stored in on-chip memory for longer than a cycle, as inference completes in a single cycle.

From this example, we see that on-chip inference NNs must be small enough to be run fast enough in the given resource constraints. These small NNs are often referred to as "edge NNs." Many edge NNs have unique characteristics that have not been considered extensively in prior work, namely:

(1) heavily quantized ($\leq$ 8 bits) fixed-point weights, and
(2) fully on-chip inference that expose weights to faults.

Some techniques from prior work translate well to edge NNs while others do not. Most prior work has studied NN faults by conducting fault injection (FI) campaigns. FI simulates faults in software or hardware and passes a number of inputs through the NN to assess performance. This is computationally expensive because an enormous amount of faulty scenarios must be simulated. Previous work has sought to limit the search space [18, 61, 78]. One method [18] exploits how a more significant bit is at least as sensitive as a less significant bit—a finding that is consistent with our own results and that we exploit too. But, this method and others find that much of the improvements in reducing the FI search space is a direct result of using float32 to represent values. The search space of float32 is large and prior work [78] trivially cuts down the space by focusing on only the eight exponent bits. A more clever method of statistically reducing the search space directly hinges on how flipping bits in float32 values results in massive changes in magnitude [61]. This big swing in magnitude does not occur with low-precision fixed-point/integer data types because the range of values that they can represent is considerably smaller, e.g., ECON-T's 6-bit weight contains only 1-integer bit, leading to a maximum difference of 2. Our work therefore seeks to fill this gap and *study how faults affect edge NNs built for on-chip inference.*

We introduce `FKeras`, a tool that assesses the sensitivity of NN weights to faults, specifically targeting edge NNs for on-chip inference.[1] `FKeras` includes fast and efficient metrics that provide a bit-level sensitivity ranking of the weights, including a novel metric based on the Hessian (second-order partial derivatives of the NN). Our fault tolerance metrics reveal under the hood how sensitive a NN is to faults, facilitating the addition of fault tolerance into the codesign problem by allowing the designer to consider how different quantized edge NNs handle faults. Furthermore, our metrics provide a way to evaluate fault tolerance alongside performance, resource usage, and power consumption, which can result in hardware accelerators that are smaller, more performant, and more resilient. For instance, the ECON-T uses triple modular redundancy (TMR) to protect the NN weights against faults [24]. TMR is effective but incurs a 200% overhead [7, 67], which is particularly costly when every resource counts in ECON-T's 4 mm^2 area budget. Many interesting design tradeoffs emerge when considering fault tolerance: Can one TMR a subset of the parameters to optimize the NN architecture in another manner? Which computation and data are the most important to protect against faults? How does quantization affect the NN's fault resilience? How do different NN architectures compare with respect to accuracy, performance, and fault tolerance?

`FKeras` facilitates fault analysis with hls4ml [30]. hls4ml targets edge ML applications with low latencies, high throughput, minimal power budgets, and low resource usage [31]. `FKeras` extends the hls4ml workflow to assess the sensitivity of NN weights to faults, perform efficient FI campaigns, and facilitate design space exploration that considers fault tolerance alongside accuracy, performance, and resource usage.

hls4ml designs that target FPGAs/ASICs must satisfy unique requirements uncommon in other architectures like GPUs and TPUs [39]. First, hls4ml implementations are highly quantized, often using unique arbitrary precision fixed-point data types in each NN operation. Second, hls4ml implementations hold most of their data on-chip, including inputs, outputs, weights, and internal state. Third, they often operate in high-radiation or safety-critical environments. For example, the radiation of the ECON-T in the LHC is approximately 1000× that of the radiation in space. Thus, understanding the potential effects of faults is crucial.

hls4ml accelerators are often heavily quantized to meet stringent performance, power, and area requirements. Quantization reduces the computational and storage costs and modifies the sensitivity of computations to faults. QKeras [20] is a tool developed by Google and the hls4ml community to handle custom hardware data types. QKeras provides drop-in replacements for NN operations, e.g., from `Dense` to `QDense`. FKeras is modeled after QKeras, providing similar replacements for NN operators (e.g., `FQDense`). `FKeras` allows designers to consider fault tolerance in the context of fixed-point computations.

`FKeras` includes several bit-level sensitivity metrics, including

(1) most significant bits first to least significant bits last (MSB $\longrightarrow$ LSB),
(2) the gradient, and
(3) the Hessian.

We introduce a new metric based on the Hessian. The Hessian (second-order partial derivatives) captures the curvature of a NN's loss landscape, providing insight into how the NN will react to perturbations to the weights. The Hessian has been shown to capture NN sensitivity and is useful at quickly quantizing a NN to mixed precision

[1] https://github.com/KastnerRG/fkeras

bitwidths [26, 27, 74–76]. FKeras efficiently calculates the Hessian, which gives a highly competitive ranking in all our considered networks.

Our findings show that individual bits matter—some bits are more important than other bits. Within a weight, the importance tends to be monotonic [18], i.e., a more significant bit is at least as sensitive as a less significant bit, though there are exceptions. We then compare the relative effect of the bits across the weights using the Hessian or gradient.

FKeras can guide FI campaigns to inject faults on the most sensitive parameters first. The sensitivity of the weights is variable; many do not lead to faulty behaviors. Thus, a campaign should focus on injecting faults into weights that are the most vulnerable to faults. We can use our Hessian sensitivity metric to determine the most sensitive weights and flip those first. Fig. 9 shows that the Hessian sensitivity metric performs substantially better at guiding the FI towards faulty behaviors than randomly or statistically injecting faults as prior FI tools and studies do [28, 44, 59, 61].

We demonstrate the value of FKeras in considering fault analysis in the design space exploration process for a NN. We perform a design space exploration on three different NN architectures for the ECON-T autoencoder. We use FKeras to assess the resilience of these different architectures to faults alongside accuracy, performance, and resource usage. We also analyze the resilience of an edge convolutional neural network (CNN) trained on CIFAR-10 [41] that operates also completely on-chip. Our results show that different-sized networks have vastly different levels of fault tolerance, e.g., a smaller, less accurate NN has more sensitive bits than a larger, more accurate NN.

FKeras is a tool that helps design fault-tolerant NNs for on-chip inference. The primary contributions of FKeras are:

- Providing bit-level weight sensitivity metrics tailored for on-chip inference edge NNs.
- Using sensitivity metrics to guide fault injection campaigns that consider the most sensitive bits first.
- Performing NN architecture design space exploration that considers fault tolerance alongside traditional optimization criteria like performance and area.

The remainder of the paper is organized as follows. Sec. 2 discusses related work. Sec. 3 introduces FKeras. Sec. 4 describes the experimental setup and assesses the fault tolerance of four different networks using FKeras. Sec. 5 concludes the paper.

2 RELATED WORK

Prior work has sought to understand NN fault tolerance and to develop techniques to protect NNs from faults, but few have considered edge NNs that are quantized and run on specialized hardware, such as FPGAs and ASICs, completely on chip [50, 52, 71]. As we review related work, we will point out some of the shortcomings of prior work when their fault analysis methods are applied to on-chip edge NNs. We then describe how FKeras addresses these shortcomings.

It is well known that NNs have many redundant parameters [33, 34]. Faults in different weights are expressed differently, especially depending on the size of the NN, as our results later show (see Fig. 11). Furthermore, the bit position of the weights is important—within a weight, there is a significant range of effects for the different bits. Our results show that the most significant bits are the most sensitive, and the least significant bits are fairly insensitive to faults, as confirmed by prior work [18]. To optimally compress quantized NNs, it is crucial to have a *bit-level* understanding of faults, especially when we want to run an edge NN entirely on chip. Thus, we develop FKeras to quantify the fault tolerance of an hls4ml edge NN for on-chip inference.

Tab. 1 compares FKeras to prior work. It categorizes FI tools and methods based on if a tool targets single or multiple bits or randomly selects bits to fault inject (Bit Target-ability), if a tool supports different data types for quantized

FI Tool/Method	Bit Target-ability	Quantization Support	FI Speedup Method	Sensitivity Metrics
Ares [59]	Random	Fixed Point/Int	No	No
TensorFI [45]	Single, Random	No	No	No
PyTorchFI [49]	Single	Integer	No	No
GoldenEye [52]	Single	Custom	No	No
enpheeph [22]	Single, Multiple, Random	Custom	GPU support	No
[19]	N/A	N/A	N/A (Heuristic)	Gradient
BinFI [18]	Single	Limited	Element-wise binary search	No
StatFI [61]	N/A	N/A	Statistical sampling	No
FKeras	Single, Multiple, Random	Fixed Point/Int	Metric-guided FI	Hessian, Gradient, MSB $\longrightarrow$ LSB

Table 1. Comparison of FKeras to prior work. If the work presents an FI method and is not a tool, then we list the Bit Target-ability and Quantization Support as not applicable (N/A).

NNs (Quantization Support), if a tool/method presents a way to speedup a FI campaign (FI Speedup Method), and if a tool/method introduces any sensitivity metrics (Sensitivity Metrics). Next, we elaborate further on prior work.

2.1 Assessing NN Resilience

Researchers have studied the resilience of NNs to identify silent data corruptions (SDCs)—soft errors that lead to incorrect NN output [44, 67]. They take three primary approaches: fault injection (FI) campaigns, heuristics, and ML modeling [50, 54, 67].

FI campaigns simulate a fault either in software [18, 44, 49, 52, 55, 56, 59] or hardware [32, 59] and run a number of inputs through the NN to assess performance. Such campaigns are computationally expensive because they must simulate an enormous amount of faulty scenarios, especially when considering the thousands to millions of parameters and operations present in an NN [67]. Researchers have developed methods to accelerate fault injection campaigns [18, 32, 50]; however, many of their evaluations are often limited to NNs that rely on off-chip memory accesses, meaning they assume the weights are fully protected by error correction codes (ECC) or parity and thus only study the effects of faults to NN activations [4, 18, 46]. FKeras instead focuses on a fault model that targets NN weights that fully reside on chip, where ECC or parity bits require extra resources, which is costly when most resources are preciously devoted to implementing the edge NN.

Researchers have also developed heuristics that measure NN sensitivity through characteristics such as the gradient [19, 48] or how errors propagate through a model [62]. Since heuristics are naturally lossy, they are less accurate than FI campaigns, but significantly cheaper to run. They trade some accuracy for speedup.

Researchers have further developed ML models to predict which bits in a NN are mostly likely to induce faulty output [16, 68, 69, 78]. These methods extract salient features of NNs relevant to fault analysis and train a machine learning model to predict the faulty parts of the NN on a relatively small amount of ground truth data. While these models are fairly accurate, they are expensive to scale up because they need to train a new ML model for each new NN they want to evaluate.

Researchers have created a fault injection method called BinFI [18], which relies on the monotonicity of the bit order, i.e., higher order bits are at least as sensitive as lower order bits, to reduce the number of faults to inject by performing a binary search within each NN output. This is an effective approach because they reduce the search space by $O(\log(n))$ per NN output where n is the number of bits representing the output. However, our results show that this assumption of monotonicity leads to false negatives, as seen in our results in Sec. 4.2.3.

Researchers have looked into statistical fault injection [61], which we call "StatFI", to reduce the FI search space of a NN's weights. However, StatFI is primarily effective in reducing the search space because it relies on how flipping bits in float32 format leads to large changes in the magnitude of the weights, as previously discussed in Sec. 1. Based on this information, StatFI selects a subset of the bits in each bit position, e.g., MSB, in each layer to randomly flip. These large changes in magnitude do not occur with low-precision fixed-point data types because the range represented is considerably smaller. Our results show that StatFI fails to identify many sensitive bits. It is not as effective as any of `FKeras`'s sensitivity metrics when considering low-precision quantized NNs for on-chip inference. This is because StatFI is selecting which bits to flip in each bit position in each layer *randomly*, whereas `FKeras`'s metrics are selecting which bits to flip using the Hessian for instance, which captures how sensitive a given NN's parameter is to faults (see Fig. 8). We present a more detailed comparison later in Sec. 4.2.3.

In response to these shortcomings, `FKeras` provides a bit-level ranking using the Hessian matrix of second partial derivatives of the loss function to identify the most crucial weights and bits. The Hessian reveals how sensitive a weight is to faults. As a result, `FKeras`'s Hessian-based bit-level ranking uses fine-grained information on a NN's sensitivity, rather than more coarse-grained approaches that rely solely on bit order monotonicity or statistical sampling that do not translate as well to quantized, on-chip edge NNs. Even more, `FKeras` presents the opportunity to combine the FI campaign with a sensitivity metric and guide the campaign to search for more sensitive bits first. `FKeras` combines the monotonicity of the bit order with its Hessian sensitivity metric to rank the most significant bits based on its Hessian sensitivity score first before doing the same for each lower order bit. Our results show this bit-level Hessian metric can identify the most sensitive parameters with high accuracy, and thus serves as a valuable guide for FI campaigns.

2.2 Optimizing NN Resilience

Prior work has taken several approaches to protect NNs and optimize said protection. Some proactive approaches, such as fault-aware training [7, 57, 58, 64, 77], prevent faults by training the weights themselves to be more resilient. Other approaches are more reactive, including DMR and TMR [24, 65], selective DMR/TMR [7, 40, 47, 50], and activation clipping [17, 36]. Activation clipping is an attractive and inexpensive option. It involves profiling the model to capture what the numerical range of the activations should be and then clipping an activation if it falls outside the range due to faulty hardware. It is particularly effective at protecting float32 NNs as the range of float32 values is very large. However, this is less effective in quantized models because quantized data types, like int8, naturally clip the activations [50]. hls4ml networks are usually heavily quantized, making activation clipping not an option. Prior work [7, 50] also combines a number of these techniques to improve error coverage while minimizing protection overhead costs.

`FKeras` can help determine how to best protect NN weights, which is especially important for edge NNs that run fully on-chip. As we show in our experiments (Sec. 4), certain bits are much more sensitive to faults than others. Protecting only those most sensitive bits can reduce the overhead of fault-tolerant mitigation. For example, our results show that it is possible to selectively perform TMR on the ECON-T weights with minimal effect on the encoder's resilience to faults. We can use the `FKeras` sensitivity metrics to find the most important bits to protect, which is especially valuable when resources are precious for on-chip inference.

3 FKERAS

NNs are often over-parameterized, and not all weights are equally important [33, 34], which indicates that some weights are more sensitive to faults than others. `FKeras` is a codesign tool for designing fault-tolerant NNs in hls4ml. It provides a *sensitivity score* that ranks the NN parameters based on their sensitivity to faults and supports modeling single- and multiple-bit fault injection campaigns on NN weights. `FKeras` can use the sensitivity score to speed up fault injection campaigns by quickly and accurately identifying the most important NN parameters. `FKeras` is also valuable for NN co-design problems for applications that require fault tolerance.

3.1 NN Sensitivity Scores

To analyze NN sensitivity, we want to understand how a NN performs under faulty conditions, e.g., bit flips in the weights. Previously, researchers have used the gradient as a metric to capture a NN's resilience to faults [19, 48]. A NN's gradient with respect to the parameters is a vector of size n, defined as

$$\frac{\partial L}{\partial \theta} \in \mathbb{R}^n \tag{1}$$

where L is the NN's loss function and θ represents the n parameters of the NN. The gradient provides information about the steepness of the loss function.

The Hessian matrix H describes the steepness and curvature, providing additional insight into the NN behavior. It is an $n{\times}n$ matrix of the second-order partial derivatives of the loss L:

$$H = \frac{\partial^2 L}{\partial \theta^2} \in \mathbb{R}^{n \times n} \tag{2}$$

The Hessian captures the local curvature of the loss function, as it shows the rate at which the gradient changes. The local curvature of the loss reflects the sensitivity of a NN's parameters [27, 76]. A steep curvature around a given parameter indicates that it is highly susceptible to noise. Perturbing this parameter even slightly will result in significant changes to the loss, implying that the model will behave worse and lead to incorrect output. Conversely, a relatively flat curvature around a given parameter indicates that it is insensitive to noise. Small perturbations to it will result in minimal changes to the loss, i.e., the model's behavior remains about the same. Since the Hessian models parameter sensitivity, researchers have relied on it to successfully quantize NNs to mixed precision [13, 27].

Despite how valuable the Hessian is, it is not commonly used because of the misconception that computing Hessian information for a large NN is infeasible, given that it requires $O(n^2)$ memory [76]. However, extracting the Hessian eigenvalues and eigenvectors takes $O(n)$ memory in $O(n)$ time using techniques from randomized numerical linear algebra (RandNLA) [5, 29, 53, 70]. The eigenvectors and eigenvalues capture the relevant Hessian information.

`FKeras` provides a Hessian-based sensitivity score for each bit of every NN weight. The sensitivity score provides a quick and accurate method to assess the fault tolerance of an NN. This allows us to speed up fault injection campaigns and perform codesign considering fault-tolerance as a constraint.

`FKeras` uses the power iteration method to compute the top k eigenvalues and eigenvectors of the Hessian in $O(n)$ time, where n is the number of parameters [76]. Based on these k eigenvalues, we compute a parameter score:

$$H' = \sum_{i=1}^{k} \lambda_i (v_i \cdot \theta) v_i \in \mathbb{R}^n \tag{3}$$

where λ_i is the ith eigenvalue, v_i is the ith eigenvector, and θ is a vector representing the model parameters (of which there are n). The parameter i sensitivity score aims to identify which parameters contribute most significantly to the Hessian by weighting it by the eigenvalue along the most sensitive direction (eigenvector). Fig. 1 visualizes how we compute our Hessian score.

We sort the parameters' most significant bits (MSBs) by the parameter sensitivity score to get a bit-wise ranking. Then, we do the same for the parameters' ith MSB until we reach the least significant bit (LSB). We sort from MSB to

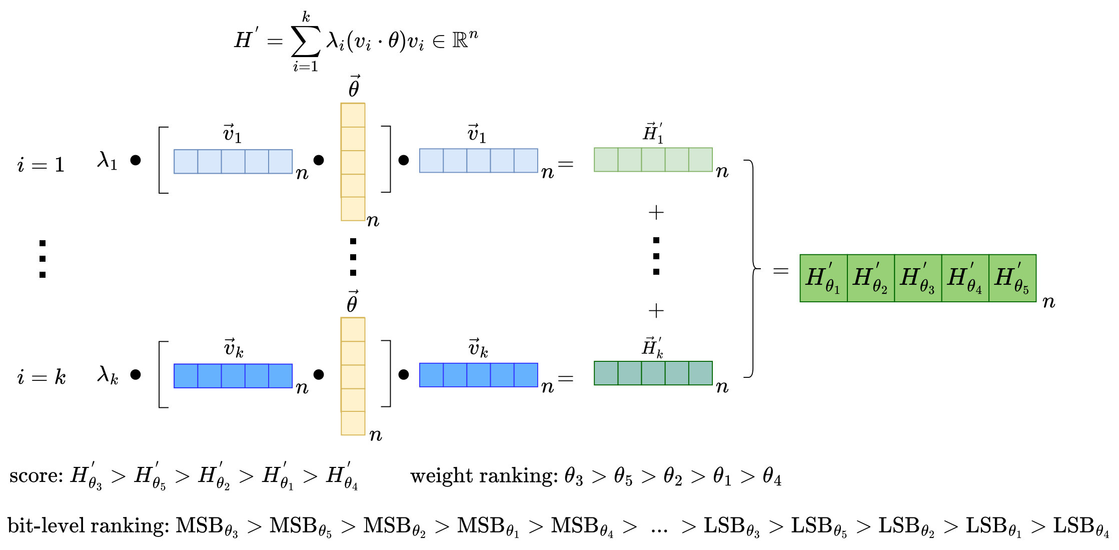

$H' = \sum_{i=1}^{k} \lambda_i (v_i \cdot \theta) v_i \in \mathbb{R}^n$
$i = 1$
λ_1
$\vec{v}_1$
$\vec{\theta}$
$\vec{H}_1'$
$i = k$
λ_k
$\vec{v}_k$
$\vec{H}_k'$
n
H'_{θ_1} H'_{θ_2} H'_{θ_3} H'_{θ_4} H'_{θ_5}
score: $H'_{\theta_3} > H'_{\theta_5} > H'_{\theta_2} > H'_{\theta_1} > H'_{\theta_4}$
weight ranking: $\theta_3 > \theta_5 > \theta_2 > \theta_1 > \theta_4$
bit-level ranking: $\mathrm{MSB}_{\theta_3} > \mathrm{MSB}_{\theta_5} > \mathrm{MSB}_{\theta_2} > \mathrm{MSB}_{\theta_1} > \mathrm{MSB}_{\theta_4} > \ldots > \mathrm{LSB}_{\theta_3} > \mathrm{LSB}_{\theta_5} > \mathrm{LSB}_{\theta_2} > \mathrm{LSB}_{\theta_1} > \mathrm{LSB}_{\theta_4}$

Fig. 1. **How to compute our Hessian sensitivity score.** We compute our Hessian sensitivity score using the top k eigenvalues and eigenvectors of the Hessian, where λ_i is the ith eigenvalue, $\vec{v_i}$ is the ith eigenvector, and $\vec{\theta}$ is a vector representing the model parameters (of which there are n). For instance, for $i = 1$, we multiply the top-1 eigenvalue λ_1 with the dot product of top-1 eigenvector $\vec{v_1}$ and the parameters $\vec{\theta}$. We then multiply this constant result element-wise with the top-1 eigenvector v_1 to get the Hessian score $\vec{H_1'}$ based on the top-1 eigenvalue and eigenvector. We then sum these top-k Hessian score vectors together to get the final Hessian score $H' \in \mathbb{R}^n$ such that H'_{θ_i} is the Hessian score for the ith parameter θ_i. We rank our weights based on this score, where a higher score means the weight has higher sensitivity to faults. Then we rank all of the MSB's by the weight sensitivity score, then the second MSB's, and so on to form our Hessian bit-level ranking.

LSB, where we consider MSBs to be the most sensitive bits because they cause the most significant perturbation in the weights of the NN when flipped (based on a twos-complement representation).

FKeras also provides a sensitivity score based on the gradient. This works in a similar manner as the Hessian, but instead uses the gradient value from Equation 1 and sorts the bits from MSB to LSB in a similar manner as the Hessian.

FKeras can compute the Hessian trace in $O(n)$ time, where n is the number of parameters, using the Hutchinson method [76]. FKeras provides the trace per layer so the user can compare the layer sensitivity. A higher trace implies that a layer is more sensitive to faults and other weight perturbations.

3.2 Fault Model

Different environments have different fault rates. For example, at the LHC, high energy physicists expect a fault to occur in their NN hardware every 15 seconds whereas in data centers, system administrators expect faults to occur once every few thousand events [25, 37]. Thus, we want to model these fault rates using a *fault model*, which describes how often a bit flip occurs.

A designer can use FKeras to perform experiments on two kinds of fault models: the single-bit flip model and the multi-bit flip model.

3.2.1 Single-Bit Flip Model. A common fault model is the single-bit flip model [32, 44, 50], which represents the case when only one bit flips at a time. The single-bit flip fault model can be applied to NN weights, activations, both, or at an even finer grain [35]. We limit our scope to only the weights of a NN, as motivated by the hardware implementations of on-chip inference for edge NNs. In particular, our fault model is motivated by the ECON-T's ASIC hardware implementation, as seen in Fig. 2. In this diagram, we see that the entire NN is implemented fully on chip, and inference completes in a single cycle. Faults in the activations only affect a single inference, whereas faults in the weights will affect billions of inferences before the weights are refreshed every four hours [24]. Thus, we focus our fault model on bit flips in the NN weights.

3.2.2 Multi-Bit Flip Model. We also consider a multi-bit flip fault model to represent high radiation conditions. This is motivated by our running example—the ECON-T autoencoder ASIC operating in the LHC HGCal [24], as seen in Fig. 2.

The multi-bit flip model for the HGCal expects ~0.06 bit flips per ECON-T per second, or one bit-flip every 15 seconds for an individual ECON-T chip. Particles go through the HGCal detector at a rate of 10^8 particles/cm^2/second [2]. However, not every particle will flip a bit. The rate at which any single ECON-T flip-flop will get hit depends on a number of factors, such as the material (silicon) and the ASIC technology node (65 nm). Moreover, the ECON-T's weights are refreshed every four hours. Thus, we define our multi-bit flip model to expect 960 bit flips when there is no fault protection in between the weight refreshes.

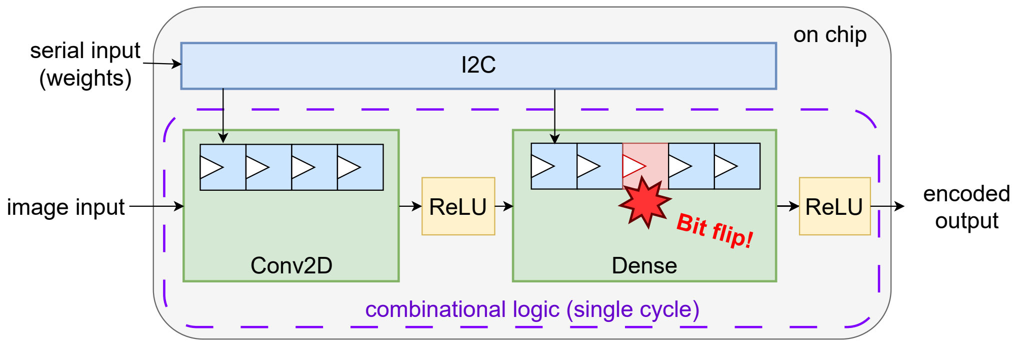


Fig. 2. **ECON-T ASIC hardware design and fault model.** The ECON-T ASIC is designed to run inference completely on chip in a single cycle. The weights are thus the most vulnerable to bit flips caused by high-energy particle strikes because they persist over billions of cycles before getting refreshed every four hours; whereas, the activations last a single cycle only.

4 EXPERIMENTAL EVALUATION

This section describes the experimental setup and results.

4.1 Experimental Setup

We demonstrate how FKeras can efficiently analyze a NN's fault sensitivity by performing experiments on CIFAR-10 [42] and an HGCal dataset. CIFAR-10 is a popular image classification dataset. The HGCal dataset contains vectors of high-energy particle collision sensor data. We use FKeras to understand the fault tolerance of four different models: (1) an edge CNN trained on CIFAR-10, specifically hls4ml's submission to the MLPerf Tiny Inference Benchmark [10], (2) a medium ECON-T NN, (3) a large ECON-T NN, and (4) a small ECON-T NN.

The three Pareto-optimal ECON-T models (Small, Medium, and Large) represent tradeoffs between model accuracy and size. All ECON-T models were trained on the HGCal dataset. We evaluate model performance using Earth mover's distance (EMD), a distance measure between two probability distributions [60]. In our case, the EMD measures the distance between the encoder's input energy readings and the decoder's outputs, respectively. Lower EMD is better and an EMD of 0 indicates the autoencoder is lossless. The three models were found using a Bayesian optimization neural architecture search. We used HAWQ-V2 [26] to quantize the large and small models (the medium model was hand-tuned from prior work). Fig. 3 provides details on the NN topology and quantization for the three ECON-T NNs.

ECON-T Medium is the model described in the paper by Di Guglielmo et al [24] - a 2D convolution layer followed by a dense layer using a 6-bit arbitrary precision fixed point data type. It balances between accuracy and model complexity. ECON-T Medium has 2 120 weights (180 for the convolution and 1 940 for the dense layer) for a total of 12 720 weight bits.

ECON-T Large has the same two-layer structure but larger convolution and dense layers. ECON-T Large uses a 5-bit arbitrary precision fixed point data type in the convolution layer and a 7-bit arbitrary precision fixed point data type in the dense layer. ECON-T Large has 800 weights in the convolution layer, 8 192 weights in the dense layer for 61 344 total weight bits.

ECON-T Small has two dense layers both using an 8-bit arbitrary precision fixed point data type. It has 1 280 weights and 10 240 total weight bits. The first dense layer is 64×16 and the second is 16×16. There are 10 240 total weight bits.

The final benchmark is the hls4ml CIFAR-10 submission to the MLPerf Tiny Inference benchmark [10]. It uses a two-stack model with no skip connections (five convolutional layers with 32, 4, 32, 32 and 4 filters, kernel size of 1, 4, 4, 4, and 4, and strides of 1, 1, 1, 4, 1, respectively). It achieves an accuracy of 83.1%.

We used `FKeras` to perform the single- and multi-bit flip fault injection campaigns. We conduct all of the fault injection campaigns on Google Cloud's `c3-highcpu-88` compute engine which utilizes an Intel Sapphire Rapids with 88 vCPU, 44 cores and 176GB of memory. We first generate oracles for the single-bit fault models by exhaustively performing single-bit flips on the weights and determining their effect. We create the single-bit flip oracle for CIFAR-10 by flipping a parameter bit and evaluating the model on 8 313 test images. This is a subset of the 10 000 images provided by the test dataset. This subset only includes the images that the CNN correctly classifies under non-faulty conditions. If flipping a bit causes the model to mispredict an image, we classify that bit as sensitive. To generate the single-bit flip oracle for the HGCal dataset, we flip a bit and evaluate the model on 20 000 validation inputs. We classify a bit as sensitive if flipping it causes the model error to exceed the average non-faulty model error.

We perform our multi-bit flip experiments on the ECON-T Medium model. The extremely high-radiation environment of the HGCal makes the ECON-T models subject to multi-bit flips. We flip 960 random parameter bits and evaluate ECON-T Medium on 20 000 validation inputs. We perform this process 14 000 times for the baseline ECON-T model, which corresponds to a 98% confidence level with 1% confidence intervals. This 14 000 sample size was determined based on a statistical model defined by [43].

4.2 Single Bit Flip Results

4.2.1 ECON-T Medium Model Resilience. We perform the first set of experiments on the ECON-T Medium Pareto autoencoder. Fig. 4 shows the fault sensitivity of the encoder's parameter bits. The first 180 weights correspond to the encoder's convolutional layer, and the remaining weights correspond to the encoder's linear layer. The bit index is the index of the bit that was flipped, where bit 0 is the sign bit, bit 1 is the integer bit, and bits 2–5 are the fractional bits. As expected, the sign bit and integer bit create the largest changes in the magnitude of the parameters, so those bit flips

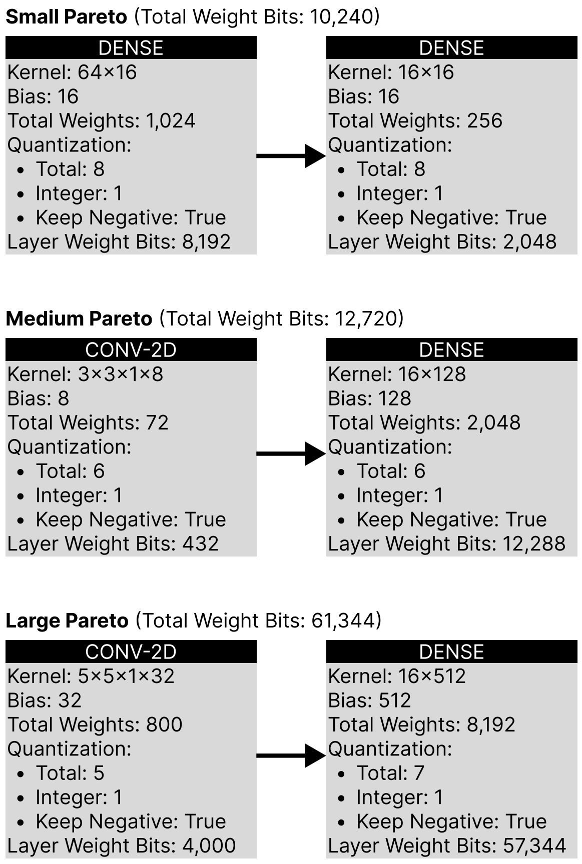


Fig. 3. A layer-by-layer overview of the three ECON-T models. Each box represents a layer in the given model. At the top of a box, we list what kind of layer it is, e.g., Dense or Conv2D. Within a box, we list details about each layer, namely the size of the kernel, the number of biases, the total number of weights, and quantization information on how each weight is quantized to fixed point according to QKeras. For example, a quantization scheme of `Total: 8, Integer: 1, Keep Negative: True` means there are a total of 8 bits, one of which represents the integer portion. When `Keep Negative` is `True`, the value is signed, and this sign bit is stolen from the fractional portion. Thus, out of a total of 8 bits, one bit represents the sign bit, one bit represents the integer part, and the remaining six bits represent the fractional part.

lead to higher EMDs. The non-faulty model has a non-zero EMD whose value is 1.10. Overall, 63.5% of the bits exceed the baseline EMD.

Not surprisingly, the largest EMD values, corresponding to most faults, occur when faults are induced on the most significant bits. The MSB (Bit 0) visually has higher EMD values than the other bits. The LSB (Bit 5) barely has any visibly discernible change from the baseline EMD value. This is not surprising, and this monotonicity has been used previously to guide fault injection campaigns [18].

The first 180 weights correspond to the convolutional layer; the remaining 1940 weights are for the dense layer. Faults in the convolutional weights generally lead to more errors than faults in the dense layer. This is especially visible in the MSB. There are a lot of weights in the dense layer where a fault does not induce any additional error, and some

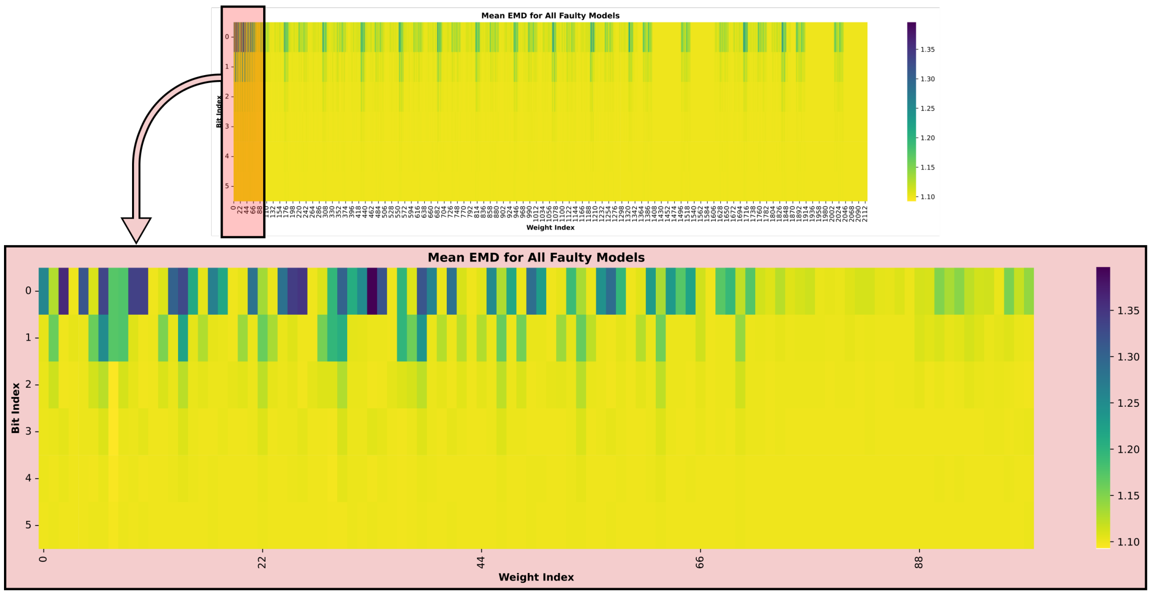


Fig. 4. The average EMD for the ECON-T medium Pareto NN under a single-bit fault model. The x-axis corresponds to the NN weight. The y-axis represents the bit index of the weight where 0 is the MSB and 5 is the LSB. Note that the top heatmap reports the average EMD over all weight bits whereas the bottom heatmap only reports the average EMD for the first 600 weight bits (NN weights 0 - 100).

in the convolutional layer. The variability of the EMD across bits of the same significance can vary greatly. For example, many of the dense layer Bit 0 weights have high EMD, but many have low EMD.

4.2.2 Sensitivity Ranking Accuracy. Our second set of experiments evaluates the ability of the Hessian and gradient to provide a bit-level sensitivity ranking. In particular, we aim to understand how well the model error (e.g., EMD) is captured by the Hessian and gradient sensitivity metrics from Sec. 3.1. We calculate these two bit-level metrics for the ECON-T Medium Pareto model and compare them with the single-bit fault EMD values from the experiments in the previous section.

Fig. 5 compares the EMD values versus the Hessian and gradient bit-level sensitivity ranking values. We separate out the rankings by bit, i.e., the first column is the MSB, and the sixth column is the LSB. These match the bit indices from Fig. 4. The first row plots the EMD (y-axis) against the Hessian ranking (x-axis) for each of the bits.

Generally speaking, we want a metric that ranks the weights that cause little change in EMD lower than those that have a higher change. Consider the Hessian MSB (bit 0) shown in the upper left plot. The EMD values of the lowest-ranked bits ranked are generally very small. After that, the EMD generally increases though there are certainly outliers. The Spearman's rank correlation coefficient $\rho = 0.508$ shows a high correlation between the Hessian ranking and the EMD. In the Hessian LSB (upper right figure), almost all bits have a very small EMD; thus, the ranking is somewhat arbitrary ($\rho = 0.098$). The major takeaway is that the more significant bits matter more. and the errors in the least significant bits have relatively little effect on this model. The first three significant bits cause the most faults, whereas faults in the least three significant bits induce little error.

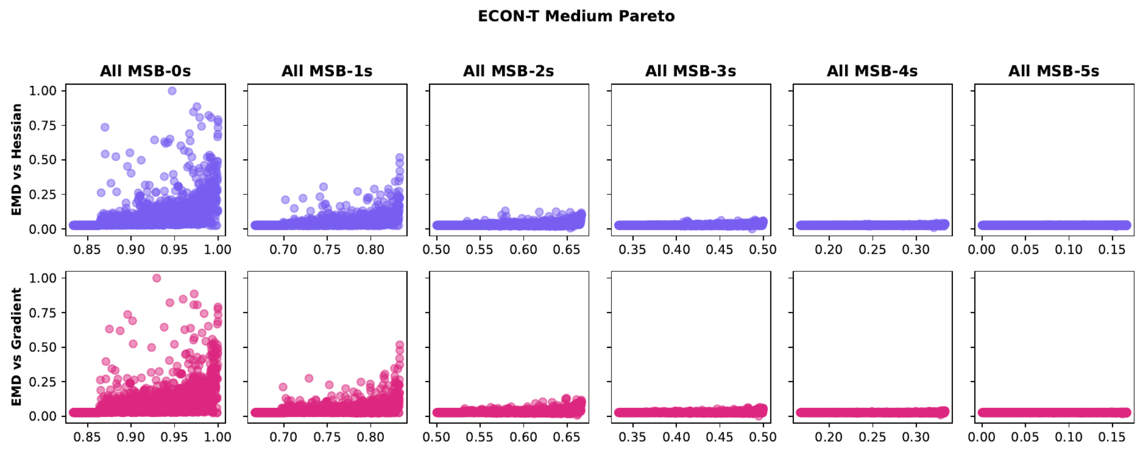


Fig. 5. A comparison of the normalized EMD (y-axis) versus the normalized Hessian and gradient sensitivity metrics (x-axis). The first column is the most significant bit and is ordered to the least significant bit in the sixth column.

The second row plots the EMD against the gradient sensitivity ranking. For this model, the Hessian performs better than the gradient in the most significant bits. The performance of the least significant bits is less important since they induce few errors.

The Hessian and gradient are measures of the loss landscape that provide valuable insight into how the NN behaves with respect to the different weights. The Hessian and gradient provide a metric to compare different weights. They do not have the ability to rank individual *bits* across those weights. Our Hessian and gradient metrics rank the most significant bits highest (Bit 0), followed by the second most significant bit (Bit 1), all the way down to the least significant bit (Bit 6 in this case). This is not ideal, and a better bit-level ranking can be achieved by mixing bits of different significance. This is clearly shown in Fig. 5. Some bit 1 EMD values are higher than the bit 0 values in the ECON-T Medium model. Some bit 2 EMD values are higher than bit 1 and bit 0, and so on. Those bit 1 variables should be ranked higher by the sensitivity metric, and both of our metrics do not allow for this. We believe that a similar approach to BinFI [18], which attempts to find the inflection point within a weight, could lead to a productive metric. We leave that as future work. However, the ranking of MSB to LSB works well as a first-order approximation. We compare the benefits in the next set of experiments.

4.2.3 Sensitivity Metric Comparison. Our next set of experiments aims to understand how different metrics perform at identifying the bits most sensitive to single-bit faults. We use four different models—three different ECON-T autoencoder models and a CIFAR-10 edge CNN, specifically hls4ml's submission to the MLPerf Tiny Inference Benchmark [10]. We compare the abilities of four metrics to rank the weights: *random*, *MSB to LSB*, *Hessian*, and *gradient*. *Random* picks a bit at random. *MSB to LSB* selects the most significant bits first, followed by the second most significant bit, all the way to the least significant bits. The bits are selected in the weight index provided by Keras after flattening a layer's weight matrix, e.g., the ECON-T Medium NN has the weight ordering shown in Figure 4. *Hessian* uses the Hessian-based sensitivity score as computed in Equation 3. *Gradient* uses the parameter's gradient value from Equation 1 and sorts the bits from MSB to LSB in a similar manner to *Hessian* (see Section 3.1).

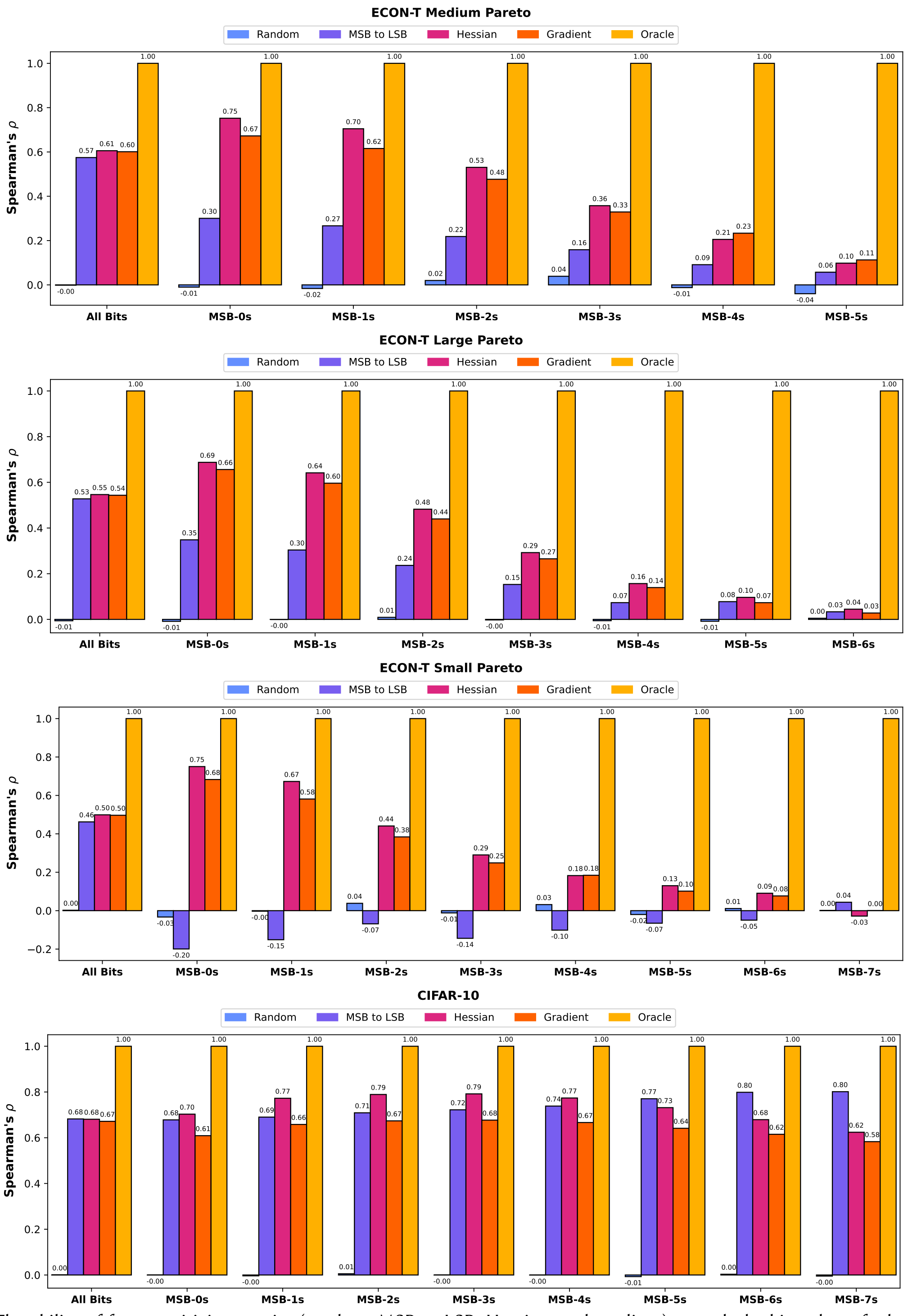


Fig. 6. The ability of four sensitivity metrics (random, MSB to LSB, Hessian, and gradient) to rank the bits whose faults induce the most errors. Spearman's rank correlation coefficient ρ is provided for all the bits in the first column and then broken down by individual bits from most to least significant.

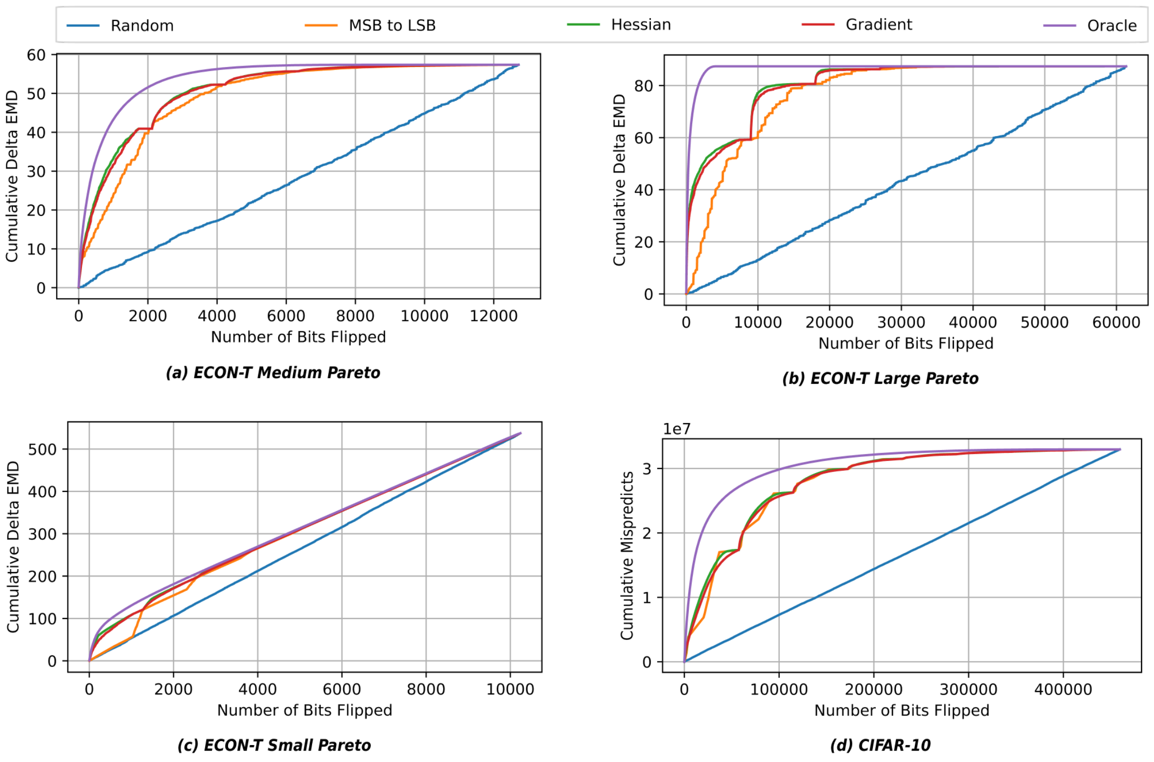


Fig. 7. The magnitude of the error vs. the number of bits flipped for the four NN models. The three ECON-T models use cumulative ΔEMD, where $\Delta\text{EMD} = |\text{EMD}_{\text{bitflip}} - \text{EMD}_{\text{non-faulty}}|$ as the error measure and the CIFAR-10 CNN uses the cumulative number of mispredicts. In both cases, larger indicates more errors. Each model plots five ranking metrics which attempt to order the NN weight bits from those to contribute most to error to those that contribute little or nothing to the error.

We compare the ability of these metrics to predict the sensitivity accurately. Fig. 6 shows the Spearman's rank correlation coefficient ρ for the four different models. The results compare the four metrics and an *Oracle* (a perfect ranking derived from the brute-force single-fault experiments) to predict the bit sensitivity. The ρ values are shown for all the bits, followed by them being broken out into the individual bits in order of most significant to least significant. As expected, the random ranking has a near zero correlation, and the oracle has a perfect correlation across all the models. The Hessian is consistently the best, especially in the more significant bits. The gradient also provides good performance close to the Hessian but clearly lower in most cases. The MSB to LSB ranking performs quite well and provides a relatively simple way to consider bit sensitivity, e.g., for a fault injection campaign.

Next, we consider the relative magnitude of the error and not just the relative ranking. Fig. 7 shows the results of an experiment that plots the cumulative error when ranking the sensitivity of the weight bits. A larger error indicates that flipping that particular bit increases the error of the overall model. The error measure depends on the model. The three ECON-T autoencoder NNs use EMD for error. Recall that EMD is a measure of error where larger indicates worse autoencoder performance. The CIFAR-10 CNN uses the number of mispredictions for the error where larger indicates more error.

Consider first Fig. 7a that plots the cumulative ΔEMD versus the number of bits flipped for the four metrics and an oracle on the ECON-T Medium Pareto NN. The oracle is the optimal or best-case ranking calculated from the brute-force single-fault experiments (e.g., from Fig. 4 for ECON-T medium). The oracle ranks the bits with the largest mean ΔEMD first. The cumulative EMD provides the difference between the faulty model EMD and the EMD of a model with no faults. The EMD for the non-faulty model is 1.100. The cumulative ΔEMD for the oracle results quickly approaches the maximum cumulative ΔEMD of 57.37. Only 63.5% (8 080/12 720) of the bits are sensitive, i.e., they have a nonzero ΔEMD. The remaining 36.5% do not affect the autoencoder EMD.

The *random* metric is the worst of the metrics showing that chance alone provides roughly an equal chance of guessing the bits that contribute most to the EMD. *MSB to LSB* performs significantly better than random. This shows that the bit order matters. The most significant bit has the lion's share of the cumulative ΔEMD (40.91 of the 57.37). The impact on ΔEMD falls quickly; weights from last few significant have little effect on the EMD. *Hessian* and *gradient* both perform better. *Hessian* does perform better at ordering the MSB weights with *Hessian* being slightly better as indicated by the separation between the two lines. In particular, *Hessian* is more accurate for the first 2 120 bits (corresponding to the weights of the most significant bit). The subsequent bits are approximately equal between *Hessian* and *Gradient*. These bits contribute less to the overall EMD and thus are overall less sensitive.

It is interesting to compare the difference between the *random* and *oracle* on the three ECON-T NNs. ECON-T Small NN (Fig. 7c) has a much smaller spread due to the fact that the model is smaller and all of the weights are more sensitive. Conversely, the spread in the large ECON-T NN (Fig. 7b) is the largest of the three. The large model has a small percentage of sensitive weights as indicated by the steep initial slope of the *Oracle*. In other words, the vast majority of the weights are insensitive to faults, which is not surprising given that the model has many more weight bits. The ECON-T large NN has 61 344 weight bits compared to ECON-T medium (12 720 weight bits) and the ECON-T small (10 240 weight bits).

CIFAR-10 is a different classification problem with a different error measure. Thus, the results are not as easily comparable as the three ECON-T NNs. Overall, *CIFAR-10* is the largest model with 459 520 weight bits. The fairly steep initial slope of the *Oracle* indicates that most of the sensitivity resides in a small number of bits. However, there is a relatively long tail, e.g., more similar to ECON-T medium Pareto NN. The relatively large separation between the *Random* and *Oracle* indicates that the bit sensitivity is not easy to predict. *Hessian* generally performs best in determining the most sensitive bits.

In Fig. 8, we compare the cumulative ΔEMD and mispredicts with state-of-the-art work in fault injection: BinFI [18] and StatFI [61]. To recap Sec. 2, BinFI performs a binary search within each value in the NN to find the bit that is the "inflection point," wherein all the bits more significant than it are considered sensitive to faults. BinFI calls this the *silent data corruption (SDC) boundary*. While BinFI applies this technique to the NN activations, we instead apply it to the NN weights according to our fault model. Since BinFI performs a binary search to find the SDC boundary, we first plot the actual bits BinFI flips during the binary search, which we call *BinFI (Actual Bits Flipped)*. The SDC boundary implies that all the bits more significant than it are also sensitive. We plot the actual bits flipped plus these implied sensitive bits as *BinFI (Actual+Implied Bits Flipped)*. BinFI does not specify the order in which to search the NN values, so we flip them in the order they appear in the NN. StatFI introduces two fault injection methods for finding the sensitive NN weight bits: data-aware and data-unaware. StatFI statistically determines the sample size of how many bits to flip per weight bit index, e.g., the MSB or second MSB, per layer. *StatFI (Data-aware)* statically measures the changes in magnitude in each weight that occur from flipping a bit to determine the sample size per weight bit index per layer. The larger the magnitude change the higher the sample size will be and vice versa. *StatFI (Data-unaware)* does not rely on

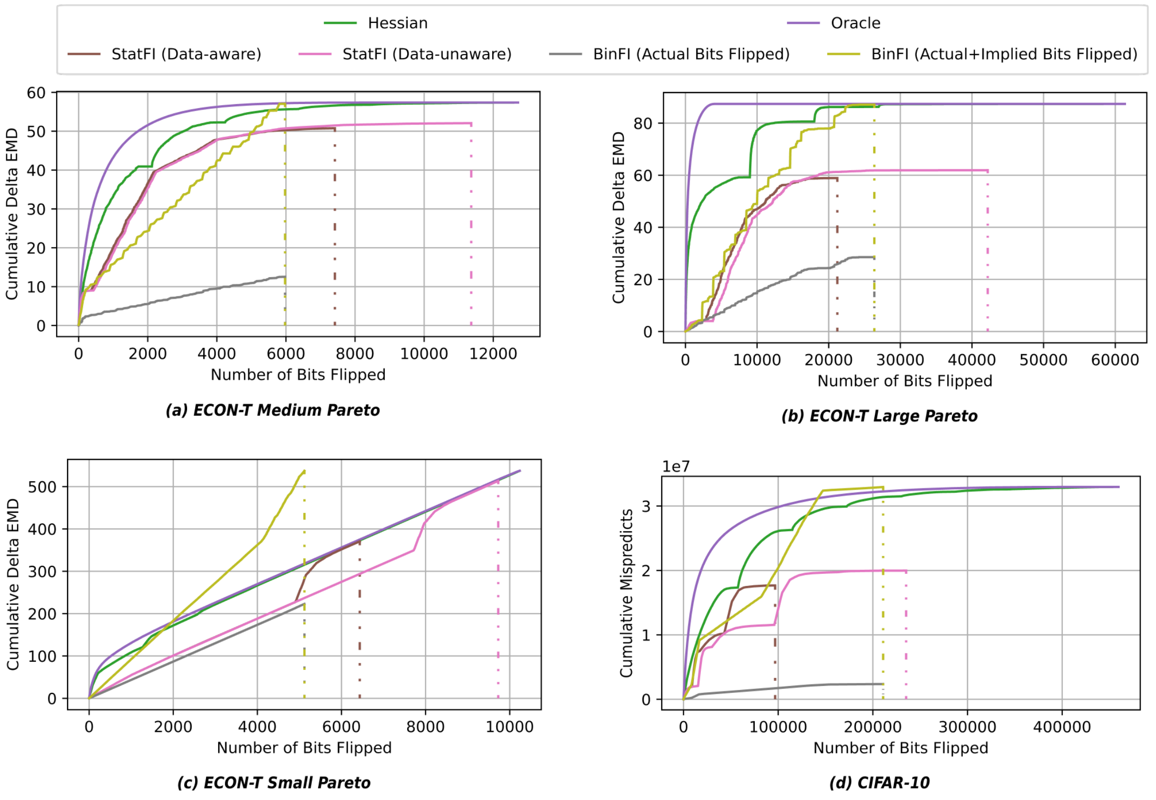


Fig. 8. Comparing the magnitude of the error vs. the number of bits flipped for the four NN models with state-of-the-art fault injection methods BinFI [18] and StatFI [61]. The three ECON-T models use cumulative ΔEMD as the error measure and the CIFAR-10 CNN uses the cumulative number of mispredicts. In both cases, larger indicates more errors. Each model plots six ranking metrics which attempt to order the NN weight bits from those to contribute most to error to those that contribute little or nothing to the error. In particular, we compare our own Hessian metric to BinFI's and StatFI's bit flips.

changes in magnitude and selects the same sample size per weight bit index per layer. StatFI randomly sample based on the determined sample size, i.e., they do not specify the order in which to flip bits. We thus order them MSB to LSB, as StatFI provides a list of bits to flip per MSB, second MSB, etc.

Let us first consider Fig. 8a and look at how BinFI compares with the Hessian and the Oracle on the ECON-T Medium Pareto NN. We find that *BinFI (Actual Bits Flipped)* is not very impressive, as expected. BinFI uses each of these bit flips to implicate more bits. As such, *BinFI (Actual+Implied Bits Flipped)* performs more impressively in the beginning; however, it begins to falter after a few hundred bit flips. This is expected because BinFI does not specify an order in which to search the weights, whereas the *Hessian* does. By only flipping around half of all of the NN weight bits though, *BinFI (Actual+Implied Bits Flipped)* identifies most of the sensitive bits, as indicated by the dashed vertical line that drops down after ~6 000 bits flipped. It fails to identify 5% of the sensitive bits (676 out of a total of 8 080). Comparing *BinFI (Actual+Implied Bits Flipped)* to our work, the *Hessian* finds more sensitive bits much sooner in its search. In fact, for the first ~5 500 bit flips, the *Hessian*'s bit flips are much more informative, as it finds the highly sensitive bits early on. Thus from Fig. 8a as well as Fig. 8b, we observe this tradeoff of the *Hessian* finding more sensitive bits earlier in the search versus the *BinFI (Actual+Implied Bits Flipped)* finding a majority of the sensitive bits earlier for the ECON-T Medium and Large Pareto models.

Fig. 8c and Fig. 8d show more complex trends. In both the ECON-T Small Pareto (Fig. 8c) and CIFAR-10 (Fig. 8d), we see that *BinFI (Actual+Implied Bits Flipped)* rises slowly, as we have seen previously, before exceeding the *oracle*. This happens because *BinFI (Actual+Implied Bits Flipped)* implies multiple bits are sensitive per bit flip whereas the *oracle* only implicates one bit based on how we plot it. Clearly, the *oracle* knows all of the sensitive bits prior to fault injection (and could reach the maximum cumulative ΔEMD/mispredicts with 0 bit flips); however, we are interested in understanding the ceiling for the*Hessian* in our *oracle*. As such, *BinFI (Actual+Implied Bits Flipped)* exceeds the best the Hessian could perform for these two models. This is likely the case because most of the bits in the ECON-T Small Pareto and CIFAR-10 models are sensitive. 100% of the ECON-T Small Pareto bits and 82.72% of the CIFAR-10 bits are sensitive. Thus, these NNs are easier tasks for BinFI—each bit flip is highly likely to find a sensitive bit. We are not necessarily finding a needle in a haystack the way we are for ECON-T Large, where only 6.5% of the bits are sensitive. The *Hessian* is clearly better in this case (Fig. 8b).

However, there is a major caveat with BinFI: the only information we receive on how sensitive the model bits are is from *BinFI (Actual Bits Flipped)*. As seen in all four charts in Fig. 8, *BinFI (Actual Bits Flipped)* reveals very little information and has the lowest cumulative ΔEMD/mispredicts out of all of the methods. We have no way of determining cumulative ΔEMD/mispredicts unless we actually flip all the bits plotted by *BinFI (Actual+Implied Bits Flipped)*. As a result, it is difficult to determine which bits are a higher priority to protect, which may be a tradeoff worth considering in the resource-constrained environments of edge NNs. Overall, BinFI performs well when the lion's share of a model's bits are sensitive and poorly when most of a model's bits are insensitive.

StatFI performs the worst in all cases in Fig. 8. Its statistical sampling is ineffective at selecting the sensitive bits in a NN. In particular, *StatFI (Data-aware)* depends on large changes in magnitude from flipping a weight bit to determine the sample size per weight bit index per layer. These large changes are more likely to occur in float32 and less likely to occur when we represent our weights with ≤8-bit fixed point precision. We therefore observe a large gap between the *StatFI (Data-aware)* line and the Hessian line in the ECON-T Medium Pareto (Fig. 8a), ECON-T Large Pareto (Fig. 8b), and CIFAR-10 (Fig. 8d) models charted. We would expect either StatFI method to work well for ECON-T Large Pareto, where few of the bits are sensitive because it was designed to find the few sensitive bits with fewer fault injections; however, StatFI randomly selects the bits to sample per weight bit index per layer, which is ineffective. For the ECON-T Small Pareto model (Fig. 8c), where 100% of the bits are sensitive, *StatFI Data-aware* fails to identify 37% of the sensitive bits, whereas *StatFI Data-unaware* fails to identify 5% of the bits. *StatFI (Data-unaware)* samples more bits, as we see in the pink line falling down after having flipped more bits than *StatFI (Data-aware)*'s brown line falling point in every case; nevertheless, it fails to find many of the bits, especially for the ECON-T Large Pareto (Fig. 8b), missing 35% of the sensitive bits, and for CIFAR-10 (Fig. 8d), missing 76% of the sensitive bits. Both the *StatFI Data-aware* and *Data-unaware* methods are not sampling cleverly enough. Since the *Hessian* captures how sensitive the NN weights are, it easily outperforms both *StatFI Data-aware* and *StatFI Data-unaware*.

Fig. 9 provides a different analysis related to the ability of the sensitivity metrics to find the top-k percentile of the sensitive bits. The *Oracle* provides a perfect ranking with respect to the bit's sensitivity. In other words, the oracle perfectly selects the top-k percentile of the bits and provides a lower bound (best-case result) for a sensitivity metric. *Oracle* will not go to 1.0 on the y-axis when a subset of the bits are insensitive to single-bit faults. For example, only 63.5% of the bits are sensitive for the ECON-T Medium Pareto NN, only 6.55% of the bits are sensitive for the ECON-T Large Pareto NN, 100% of the bits are sensitive for the ECON-T Small Pareto NN, and 82.72% of the bits are sensitive for

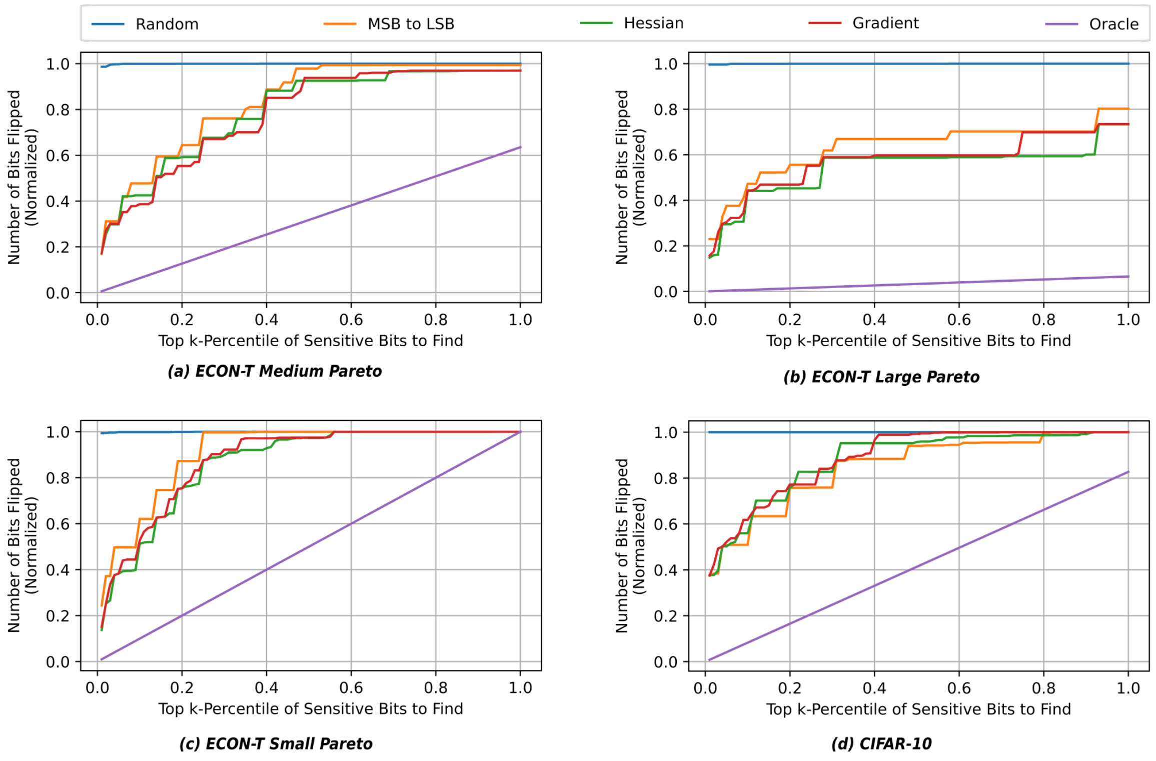


Fig. 9. The ability of the sensitivity metrics to find the top k-percentile of the sensitive bits across the four NN models. The *oracle* provides a perfect ordering of the bits and thus provides a best-case result (lower bound).

the CIFAR-10 NN. *Random* provides the other extreme as the top-k bits are randomly distributed through its ranking, and thus the entire ranking must be enumerated to find the top-k bits for all but the smallest values of k.

For the ECON-T NNs, *MSB to LSB* performs the worst of the other metrics. Overall, *Hessian* is better than *Gradient* for the ECON-T Large and Small Pareto NNs. *Gradient* is overall generally lower (better) than *Hessian* for the ECON-T Medium Pareto model.

The CIFAR-10 classification task is interesting in that *MSB to LSB* performs quite well overall, while it is the worst sensitivity metric for the ECON-T NNs. The CIFAR-10 NN is a two-stack ResNet model with five convolutional layers [10]. This is fundamentally different than the ECON-T models, which have only two layers, especially for ECON-T Small which consists of only dense layers.

We then compare the top-k percentile performance of the *Hessian* and the *oracle* with BinFI [18] and StatFI [61] in Fig. 10. We first compare with BinFI. Let us first focus on the ECON-T medium model in Fig. 10a. To find the top-1 percentile of the sensitive bits, the *Hessian* only needs to flip ~18% of the bits, whereas *BinFI (Actual+Implied Bits Flipped)* must flip ~41% of the bits, taking longer to find the most sensitive bits. BinFI then drops to 0 after the top-15th percentile because it produces false negatives, i.e., it does not find all of the sensitive bits. The top-k percentile requirement is stringent. If a method fails to identify even a single bit in the top-k percentile, then we say this method has failed because there is no number of bits to flip according to that method that will find all of the top-k sensitive bits. We can

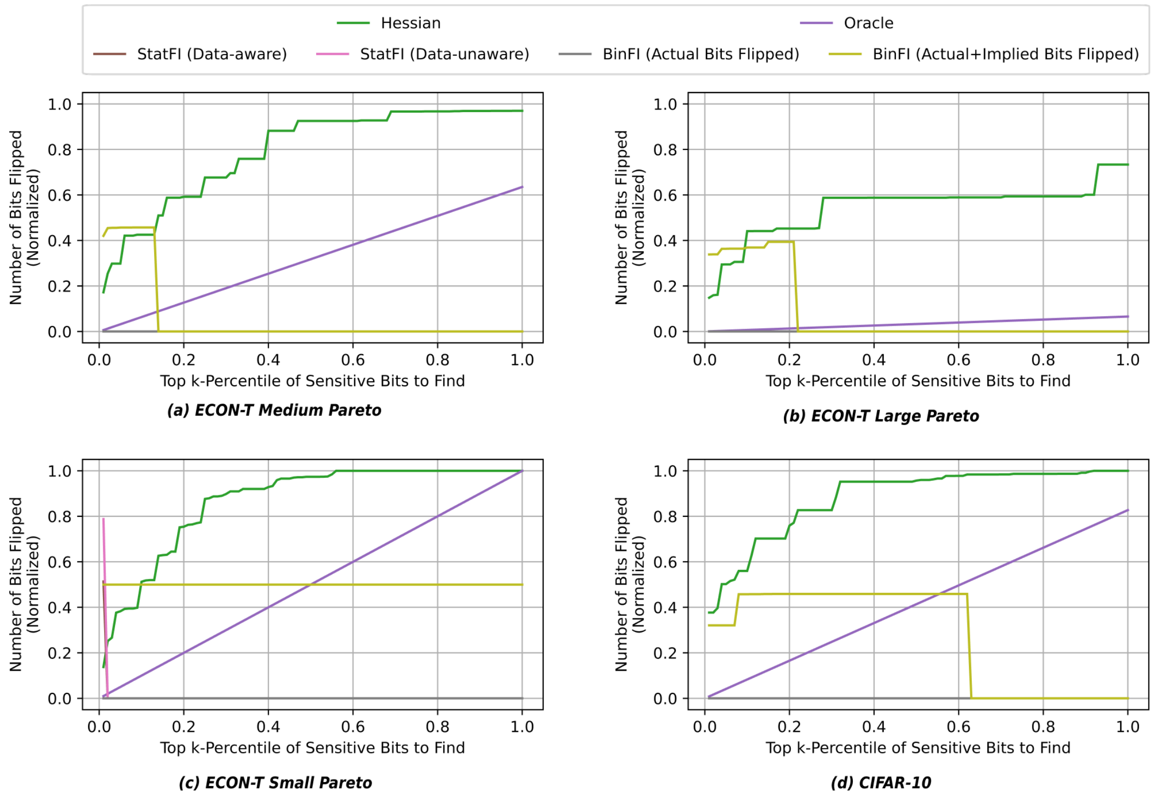


Fig. 10. Comparing the ability of the sensitivity metrics to find the top k-percentile of the sensitive bits across the four NN models with state-of-the-art fault injection methods BinFI [18] and StatFI [61]. . The *oracle* provides a perfect ordering of the bits and thus provides a best-case result (lower bound).

thus infer from Fig. 10a that *BinFI (Actual+Implied Bits Flipped)* fails to identify bits beyond the top 15th percentile, e..g, it misses bits that are quite sensitive (such as in the 20th percentile). The *Hessian* provides both weight-level and bit-level sensitivity information to rank weight bits, whereas BinFI only provides bit-level sensitivity information per weight without any way of ranking the weights. Therefore, the *Hessian* is significantly more efficient than BinFI at finding the most sensitive bits because it captures more sensitivity information. For the ECON-T Large Pareto (Fig. 10b), the Hessian outperforms *BinFI (Actual+Implied Bits Flipped)* for the top 10th percentile before *BinFI (Actual+Implied Bits Flipped)* exceeds the *Hessian* up until the top 20th percentile when it falls to 0—once again due to its failure to find sensitive bits. For ECON-T Small Pareto, the *Hessian* is the closest to the *oracle* up until the top 15th percentile when *BinFI (Actual+Implied Bits Flipped)* is the better method at finding top sensitive bits, eventually exceeding the *oracle* past the top 50th percentile. As previously discussed, this is due to *BinFI (Actual+Implied Bits Flipped)*'s implicating multiple bits as sensitive per bit flip, whereas the *oracle* only implicates one bit per bit flip. Since all the bits in ECON-T Small Pareto are sensitive, *BinFI (Actual+Implied Bits Flipped)* always succeeds in finding a sensitive bit. Moreover it only needs to flip about half of the bits to identify all of the sensitive bits. This is due to the easy nature of this task, i.e., when most if not all of the bits are sensitive. We see a similar pattern for the CIFAR-10 model where *BinFI (Actual+Implied Bits*

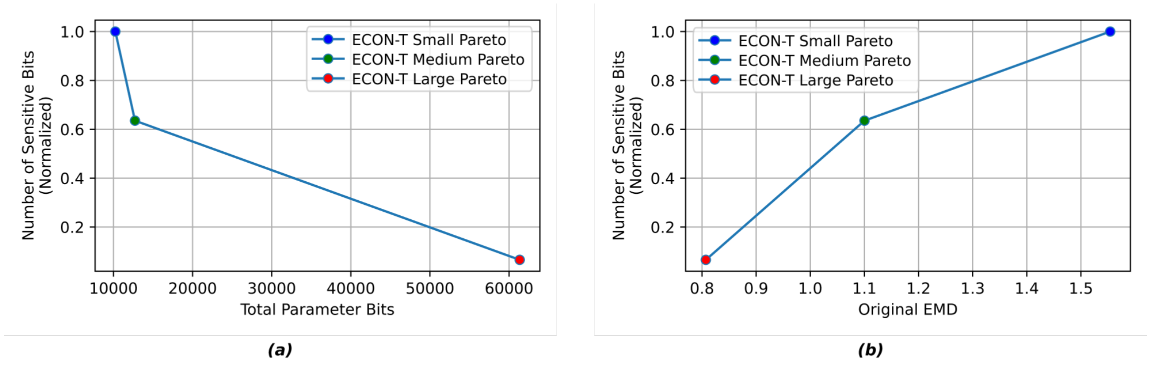
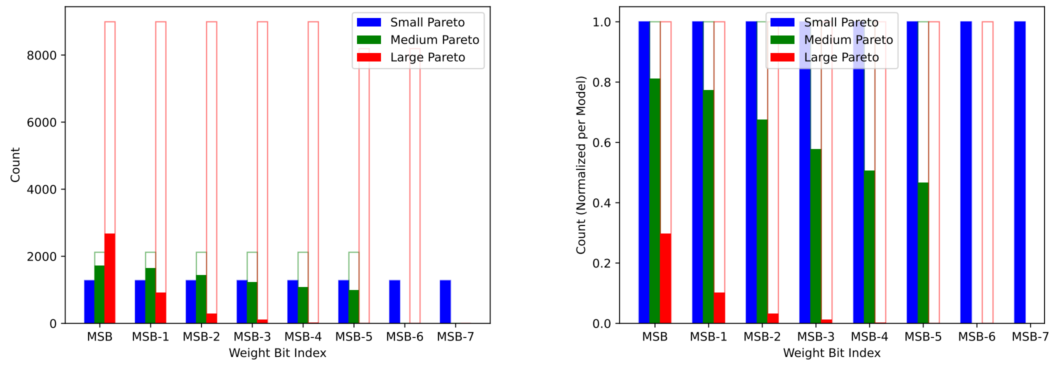


Fig. 11. Part a) As model size increases, the percentage of sensitive bits in the model decreases. Part b) As *EMD* increases, the percentage of sensitive bits in the model increases.



(a) Total Sensitive Bit Count Distribution.

(b) Normalized Sensitive Bit Count Distribution.

Fig. 12. Distribution of the sensitive bits related to bit position for the three ECON-T NNs.

Flipped) outperforms the *oracle* for a bit around the top 60th percentile before it falls to 0 due to its failure to identify all of the sensitive bits beyond the 60th percentile. Note that *BinFI (Actual Bits Flipped)* always lies at 0 for all four NNs because it is primarily flipping bits to implicate other bits in the model and is thus not good at flipping the most sensitive bits first.

We then compare with *StatFI (Data-aware)* and *StatFI (Data-unaware)*. For all four models, both StatFI methods fail to identify any top-k percentile sensitive bits, so all StatFI lines lie at 0, except for *StatFI (Data-unaware)* for the ECON-T Small Pareto model (Fig. 10c) which can find the top-1st percentile by flipping 80% of the bits before immediately falling to 0. Beyond this instance, no number of bits flipped according to StatFI will find some top-k percentile of the sensitive bits.

Next, we summarize the relationship between model size and the sensitivity of its weights. Fig. 11a plots the number of sensitive bits versus the total number of bits for the three ECON-T NNs. All of the bits in the ECON-T Small Pareto model are sensitive. As the model size increases, the number of sensitive bits decreases. The ECON-T Large Pareto model has only 6.55% of its bits sensitive to single-bit faults. Fig. 11b show the same three ECON-T models with respect to the EMD (error) of the non-faulty model. The ECON-T Large Pareto model has the smallest EMD (0.807), which is expected given that it is more complex. Reducing the model size increases the EMD (decreasing its performance).

Fig. 12 breaks out the number of sensitive bits according to their relative bit position in the weight from the MSB to the LSB. All models are quantized to a fixed-point representation such that MSB is a sign bit followed by 1–3 integer bits and some fractional bits remaining. Note that each model has a different quantization, with ECON-T Small Pareto having 8-bit weights, ECON-T Medium Pareto having 6-bit weights, and ECON-T Large Pareto having both 5-bit and 7-bit weights. In the ECON-T Small Pareto NN, all the bits are sensitive; thus the sensitive bits are equally distributed across all bit indices. 63.5% of the bits are sensitive in the ECON-T Medium Pareto NN. The sensitive bits are relatively equally distributed across each bit index though more reside in the MSB and MSB-1 bit indices. In the ECON-T Large Pareto NN, only a tiny fraction (6.5%) of the bits are sensitive. The sensitive bits are clustered in the first 3 MSBs out of (at most) 7 bits. Fig. 12b shows that when there are sensitive bits in the model, the majority of them reside in the most significant bits. Moreover, as model size increases, the percentage of sensitive bits decreases (as shown in Fig. 11a).

4.3 Multiple Bit Flip Results

We also aim to understand how the NNs respond to multiple-bit faults. Unlike the previous experiments, where we flip only one bit at a given time, we assume multiple bit flips are possible. Such scenarios are more likely in high radiation environments as is experienced by the ECON-T ASICs in the LHC. In particular, we want to understand the resilence of using protection mechanisms, e.g., using TMR on the NN weights as is done in the LHC [24].

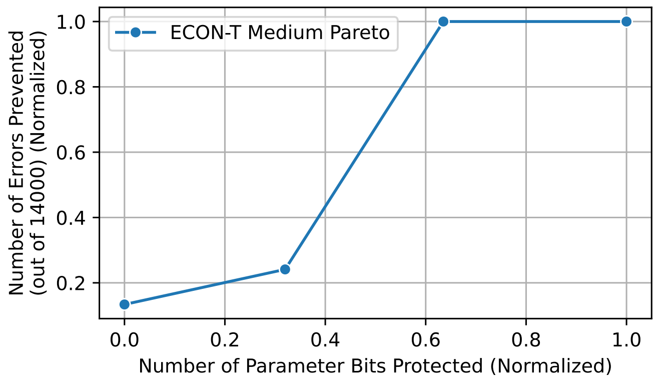


Fig. 13. The effect of bit protection on errors for the ECON-T medium Pareto NN.

For this experiment, we assume a multi-bit flip fault model with a variable amount (0%, 32%, 63.5%, and 100%) of protection for the parameter bits. For the 100% protection level, all parameter bits are protected, which means that none of the attempted bit flips will occur. For the 63.5% protection level, we protect all of the sensitive bits identified under the single-bit flip fault model. For the 32% protection level, we protect the top 50% of the sensitive bits identified under the single-bit flip fault model. For the 0% protection level, we do not protect any parameter bits. Fig. 13 shows that by protecting all of the sensitive bits identified by the single-bit flip fault model, we can prevent all of the errors induced by the simultaneous multiple bit flips. Even without protection, a small percentage (13.4%) of the multi-bit flip errors are prevented.

5 CONCLUSION

We develop FKeras as a tool to assess the fault tolerance of edge neural networks that run inference fully on chip. FKeras provides several bit-level metrics that can quickly identify NN weight bits that are most sensitive to faults. We use FKeras to study four different NN models—three Pareto-optimal models for an autoencoder hardware in the CERN Large Hadron Collider and an edge NN that performs CIFAR-10 image classification. We show that the sensitivity of the bits varies greatly across weights and that the Hessian provides a good weight sensitivity ranking. Additionally, our results indicate that the sensitivity of different bits within a weight can vary dramatically. Even more, we found that larger, more accurate NNs have signficantly fewer sensitive bits compared with smaller, less accurate NNs. This raises some architectural codesign tradeoffs: should we design hardware for a larger, more accurate NN that requires less protection or a smaller, less accurate NN that requires full protection? How much more resilience can a larger NN provide at the expense of resource efficiency? FKeras provides valuable insights for addressing these codesign tradeoffs to design fault-tolerant, quantized NNs for hardware.

ACKNOWLEDGMENTS

The authors thank the anonymous referees for their valuable comments and helpful suggestions. The authors would also like to thank Isaac Adeleke and Christopher Zutler for their generous input. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-2038238. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

This work is supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under the "Real-time Data Reduction Codesign at the Extreme Edge for Science" Project (DE-FOA-0002501). JD is also supported by the DOE, Office of Science, Office of High Energy Physics Early Career Research program under Grant No. DE-SC0021187, and the U.S. National Science Foundation (NSF) Harnessing the Data Revolution (HDR) Institute for Accelerating AI Algorithms for Data Driven Discovery (A3D3) under Cooperative Agreement No. OAC-2117997. NT is also supported by the DOE Early Career Research program under Award No. DE-0000247070.

REFERENCES

[1] Qeethara Kadhim Al-Shayea. 2011. Artificial neural networks in medical diagnosis. *International Journal of Computer Science Issues* 8, 2 (2011), 150–154.

[2] Rubén García Alía, Markus Brugger, Francesco Cerutti, Salvatore Danzeca, Alfredo Ferrari, Simone Gilardoni, Yacine Kadi, Maria Kastriotou, Anton Lechner, Corinna Martinella, et al. 2017. LHC and HL-LHC: Present and future radiation environment in the high-luminosity collision points and RHA implications. *IEEE Transactions on Nuclear Science* 65, 1 (2017), 448–456.

[3] Syed Muhammad Anwar, Muhammad Majid, Adnan Qayyum, Muhammad Awais, Majdi Alnowami, and Muhammad Khurram Khan. 2018. Medical image analysis using convolutional neural networks: a review. *Journal of medical systems* 42 (2018), 1–13.

[4] 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.

[5] Haim Avron and Sivan Toledo. 2011. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)* 58, 2 (2011), 1–34.

[6] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597* (2021).

[7] Timoteo García Bertoa et al. 2023. Fault Tolerant Neural Network Accelerators with Selective TMR. *IEEE Des. Test* 40, 2 (2023), 67. https://doi.org/10.1109/MDAT.2022.3174181

[8] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[9] Shekhar Borkar. 2005. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Ieee Micro* 25, 6 (2005), 10–16.

[10] Hendrik Borras, Giuseppe Di Guglielmo, Javier Duarte, Nicolò Ghielmetti, Ben Hawks, Scott Hauck, Shih-Chieh Hsu, Ryan Kastner, Jason Liang, Andres Meza, et al. 2022. Open-source FPGA-ML codesign for the MLPerf Tiny Benchmark. *arXiv preprint arXiv:2206.11791* (2022).

[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[12] Simon Burton, Lydia Gauerhof, and Christian Heinzemann. 2017. Making the case for safety of machine learning in highly automated driving. In *Computer Safety, Reliability, and Security: SAFECOMP 2017 Workshops, ASSURE, DECSoS, SASSUR, TELERISE, and TIPS, Trento, Italy, September 12, 2017, Proceedings 36*. Springer, 5–16.

[13] Javier Campos, Zhen Dong, Javier Duarte, Amir Gholami, Michael W Mahoney, Jovan Mitrevski, and Nhan Tran. 2023. End-to-end codesign of Hessian-aware quantized neural networks for FPGAs and ASICs. *arXiv preprint arXiv:2304.06745* (2023).

[14] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. 2019. Machine learning and the physical sciences. *Reviews of Modern Physics* 91, 4 (2019), 045002.

[15] S Chatrchyan, G Hmayakyan, V Khachatryan, AM Sirunyan, W Adam, T Bauer, T Bergauer, H Bergauer, M Dragicevic, J Eroe, et al. 2008. The CMS experiment at the CERN LHC. *Journal of instrumentation* 3 (2008).

[16] Arjun Chaudhuri, Ching-Yuan Chen, Jonti Talukdar, Siddarth Madala, Abhishek Kumar Dubey, and Krishnendu Chakrabarty. 2021. Efficient fault-criticality analysis for AI accelerators using a neural twin. In *2021 IEEE International Test Conference (ITC)*. IEEE, 73–82.

[17] 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.

[18] Zitao Chen, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2019. Binfi: An efficient fault injector for safety-critical machine learning systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–23.

[19] Wonseok Choi, Dongyeob Shin, Jongsun Park, and Swaroop Ghosh. 2019. Sensitivity based error resilient techniques for energy efficient deep neural network accelerators. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[20] Claudionor N. Coelho, Aki Kuusela, Shan Li, Hao Zhuang, Thea Aarrestad, Vladimir Loncar, Jennifer Ngadiuba, Maurizio Pierini, Adrian Alan Pol, and Sioni Summers. 2021. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nature Mach. Intell.* 3 (2021), 675–686. https://doi.org/10.1038/s42256-021-00356-5 arXiv:2006.10159 [physics.ins-det]

[21] CMS collaboration et al. 2017. The phase-2 upgrade of the CMS endcap calorimeter. *CMS Technical Design Report CERN-LHCC-2017-023. CMS-TDR-019, CERN* (2017).

[22] Alessio Colucci, Andreas Steininger, and Muhammad Shafique. 2022. enpheeph: A fault injection framework for spiking and compressed deep neural networks. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 5155–5162.

[23] Allison McCarn Deiana, Nhan Tran, Joshua Agar, Michaela Blott, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Scott Hauck, Mia Liu, Mark S Neubauer, et al. 2022. Applications and techniques for fast machine learning in science. *Frontiers in big Data* 5 (2022), 787421.

[24] Giuseppe Di Guglielmo, Farah Fahim, Christian Herwig, Manuel Blanco Valentin, Javier Duarte, Cristian Gingu, Philip Harris, James Hirschauer, Martin Kwok, Vladimir Loncar, et al. 2021. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. *IEEE Transactions on Nuclear Science* 68, 8 (2021), 2179–2186.

[25] Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. 2021. Silent data corruptions at scale. *arXiv preprint arXiv:2102.11245* (2021).

[26] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems* 33 (2020), 18518–18529.

[27] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 293–302.

[28] 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.

[29] Petros Drineas and Michael W Mahoney. 2018. Lectures on randomized numerical linear algebra. *The Mathematics of Data* 25, 1 (2018).

[30] Javier Duarte, Song Han, Philip Harris, Sergo Jindariani, Edward Kreinar, Benjamin Kreis, Jennifer Ngadiuba, Maurizio Pierini, Ryan Rivera, Nhan Tran, et al. 2018. Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation* 13, 07 (2018), P07027.

[31] Javier Duarte, Nhan Tran, Ben Hawks, Christian Herwig, Jules Muhizi, Shvetank Prakash, and Vijay Janapa Reddi. 2022. FastML Science Benchmarks: Accelerating Real-Time Scientific Edge Machine Learning. *arXiv preprint arXiv:2207.07958* (2022).

[32] Giulio Gambardella, Johannes Kappauf, Michaela Blott, Christoph Doehring, Martin Kumm, Peter Zipf, and Kees Vissers. 2019. Efficient error-tolerant quantized neural network accelerators. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 1–6.

[33] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630* (2021).

[34] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[35] Yi He, Prasanna Balaprakash, and Yanjing Li. 2020. Fidelity: Efficient resilience analysis framework for deep learning accelerators. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 270–281.

[36] Le-Ha Hoang, Muhammad Abdullah Hanif, and Muhammad Shafique. 2020. Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1241–1246.

[37] Peter H Hochschild, Paul Turner, Jeffrey C Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E Culler, and Amin Vahdat. 2021. Cores that don't count. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 9–16.

[38] IEEE 2008. *Intermittent faults and effects on reliability of integrated circuits*. IEEE.

[39] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[40] Navid Khoshavi, Arman Roohi, Connor Broyles, Saman Sargolzaei, Yu Bi, and David Z Pan. 2020. Shieldenn: Online accelerated framework for fault-tolerant deep neural network architectures. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[41] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. *Tech Report* (2009).

[42] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[43] Régis Leveugle, A Calvez, Paolo Maistri, and Pierre Vanhauwaert. 2009. Statistical fault injection: Quantified error and confidence. In *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 502–506.

[44] 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.

[45] Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. 2018. Tensorfi: A configurable fault injector for tensorflow applications. In *2018 IEEE International symposium on software reliability engineering workshops (ISSREW)*. IEEE, 313–320.

[46] Guanpeng Li, Karthik Pattabiraman, Siva Kumar Sastry Hari, Michael Sullivan, and Timothy Tsai. 2018. Modeling soft-error propagation in programs. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 27–38.

[47] Fabiano Libano, Brittany Wilson, J Anderson, Michael J Wirthlin, Carlo Cazzaniga, Christopher Frost, and Paolo Rech. 2018. Selective hardening for neural networks in FPGAs. *IEEE Transactions on Nuclear Science* 66, 1 (2018), 216–222.

[48] Abdulrahman Mahmoud et al. 2020. HarDNN: Feature map vulnerability evaluation in CNNs. In *1st Workshop on Secure and Resilient Autonomy (SARA) at MLSys 2020*. arXiv:2002.09786 [cs.LG]

[49] 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.

[50] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Christopher W Fletcher, Sarita V Adve, Charbel Sakr, Naresh R Shanbhag, Pavlo Molchanov, Michael B Sullivan, Timothy Tsai, and Stephen W Keckler. 2021. Optimizing Selective Protection for CNN Resilience.. In *ISSRE*. 127–138.

[51] Abdulrahman Mahmoud, Siva Kumar Sastry Hari, Michael B Sullivan, Timothy Tsai, and Stephen W Keckler. 2018. Optimizing software-directed instruction replication for gpu error detection. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 842–854.

[52] Abdulrahman Mahmoud, Thierry Tambe, Tarek Aloui, David Brooks, and Gu-Yeon Wei. 2022. GoldenEye: A Platform for Evaluating Emerging Numerical Data Formats in DNN Accelerators. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 206–214.

[53] Michael W Mahoney et al. 2011. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning* 3, 2 (2011), 123–224.

[54] Sparsh Mittal. 2020. A survey on modeling and improving reliability of DNN algorithms and accelerators. *Journal of Systems Architecture* 104 (2020), 101689.

[55] Niranjhana Narayanan, Zitao Chen, Bo Fang, Guanpeng Li, Karthik Pattabiraman, and Nathan Debardeleben. 2022. Fault Injection for TensorFlow Applications. *IEEE Transactions on Dependable and Secure Computing* (2022).

[56] Mohamed A Neggaz, Ihsen Alouani, Smail Niar, and Fadi Kurdahi. 2019. Are cnns reliable enough for critical applications? an exploratory study. *IEEE Design & Test* 37, 2 (2019), 76–83.

[57] Elbruz Ozen and Alex Orailoglu. 2020. Boosting bit-error resilience of DNN accelerators through median feature selection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3250–3262.

[58] Elbruz Ozen and Alex Orailoglu. 2022. Architecting Decentralization and Customizability in DNN Accelerators for Hardware Defect Adaptation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 3934–3945.

[59] 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.

[60] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The Earth Mover's Distance as a Metric for Image Retrieval. *Int. J. Comput. Vis.* 40 (2000), 99. https://doi.org/10.1023/A:1026543900054

[61] A Ruospo, G Gavarini, C De Sio, J Guerrero, L Sterpone, M Sonza Reorda, E Sanchez, R Mariani, J Aribido, and J Athavale. 2023. Assessing convolutional neural networks reliability through statistical fault injections. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[62] Christoph Schorn, Andre Guntoro, and Gerd Ascheid. 2018. Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 979–984.

[63] Nida Shahid, Tim Rappon, and Whitney Berta. 2019. Applications of artificial neural networks in health care organizational decision-making: A scoping review. *PloS one* 14, 2 (2019), e0212356.

[64] David Stutz, Nandhini Chandramoorthy, Matthias Hein, and Bernt Schiele. 2021. Bit error robustness for energy-efficient dnn accelerators. *Proceedings of Machine Learning and Systems* 3 (2021), 569–598.

[65] Emil Talpes, Debjit Das Sarma, Ganesh Venkataramanan, Peter Bannon, Bill McGee, Benjamin Floering, Ankit Jalote, Christopher Hsiong, Sahil Arora, Atchyuth Gorti, et al. 2020. Compute solution for tesla's full self-driving computer. *IEEE Micro* 40, 2 (2020), 25–35.

[66] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.

[67] Cesar Torres-Huitzil and Bernard Girau. 2017. Fault and error tolerance in neural networks: A review. *IEEE Access* 5 (2017), 17322–17341.

[68] Marcello Traiola, Angeliki Kritikakou, and Olivier Sentieys. 2023. harDNNing: a machine-learning-based framework for fault tolerance assessment and protection of DNNs. In *ETS 2023-IEEE European Test Symposium*.

[69] Marcello Traiola, Angeliki Kritikakou, and Olivier Sentieys. 2023. A machine-learning-guided framework for fault-tolerant DNNs. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–2.

[70] Shashanka Ubaru, Jie Chen, and Yousef Saad. 2017. Fast estimation of tr(f(A)) via stochastic Lanczos quadrature. *SIAM J. Matrix Anal. Appl.* 38, 4 (2017), 1075–1099.

[71] Zishen Wan, Aqeel Anwar, Abdulrahman Mahmoud, Tianyu Jia, Yu-Shun Hsiao, Vijay Janapa Reddi, and Arijit Raychowdhury. 2022. Frl-fi: Transient fault analysis for federated reinforcement learning-based navigation systems. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 430–435.

[72] Yumou Wei, Ryan F Forelli, Chris Hansen, Jeffrey P Levesque, Nhan Tran, Joshua C Agar, Giuseppe Di Guglielmo, Michael E Mauel, and Gerald A Navratil. 2023. *Low latency optical-based mode tracking with machine learning deployed on FPGAs on a tokamak.* Technical Report. Fermi National Accelerator Laboratory (FNAL), Batavia, IL (United States).

[73] Olivia Weng, Alexander Redding, Nhan Tran, Javier Mauricio Duarte, and Ryan Kastner. 2024. Architectural Implications of Neural Network Inference for High Data-Rate, Low-Latency Scientific Applications. arXiv:2403.08980 [cs.LG]

[74] Yaoqing Yang, Liam Hodgkinson, Ryan Theisen, Joe Zou, Joseph E Gonzalez, Kannan Ramchandran, and Michael W Mahoney. 2021. Taxonomizing local versus global structure in neural network loss landscapes. *Advances in Neural Information Processing Systems* 34 (2021), 18722–18733.

[75] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. 2021. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*. PMLR, 11875–11886.

[76] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. 2020. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE international conference on big data (Big data)*. IEEE, 581–590.

[77] Ussama Zahid, Giulio Gambardella, Nicholas J Fraser, Michaela Blott, and Kees Vissers. 2020. FAT: Training neural networks for reliable inference under hardware faults. In *2020 IEEE International Test Conference (ITC)*. IEEE, 1–10.

[78] Yangchao Zhang, Hiroaki Itsuji, Takumi Uezono, Tadanobu Toba, and Masanori Hashimoto. 2022. Estimating vulnerability of all model parameters in dnn with a small number of fault injections. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 60–63.

Received 28 September 2023; revised 15 March 2024; accepted 29 April 2024