FKeras: A Sensitivity Analysis Tool for Edge Neural Networks

Olivia Weng
Andres Meza
oweng@ucsd.edu
anmeza@ucsd.edu
University of California San Diego
USA

Quinlan Bock
Benjamin Hawks
Fermi National Accelerator Laboratory
USA

Javier Campos
Nhan Tran
Fermi National Accelerator Laboratory
USA

Javier Mauricio Duarte
Ryan Kastner
University of California San Diego
USA

ABSTRACT

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. 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 faulty 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 network's fault tolerance to consider alongside its accuracy, performance, and resource consumption. The results show that the different architectures have vastly differing resilience to faults.

1 INTRODUCTION

Machine learning (ML) is increasingly used in safety-critical applications, including autonomous vehicles [6, 9, 39], healthcare [2, 3, 38], and scientific experiments [11, 17, 24]. In these domains, the ML computation must act reliably in the face of errors. Soft errors are a common source of unreliability [7, 28], 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.

Consider the CERN Large Hadron Collider (LHC) Compact Muon Solenoid (CMS) experiment [12], which runs particle collision experiments that generate data rates of ~40 TB/s. With such a high rate, it is infeasible to not only store petabytes of data per minute but also to determine which data is relevant and worth further investigation. To reduce data rates, LHC physicists plan to deploy tens of thousands of encap concentrator (ECON-T) ASICs [18], each running a neural network (NN) encoder, to compress experimental data from the high-granularity endcap calorimeter (HGCAL) [16] into a smaller format for easy filtering in the trigger system. The ECON-T autoencoder hardware must accept new input data at 40 MHz and complete processing in 50 ns [18]. The core computation of the ECON-T is a two-layer neural network—a 2D convolution followed by a dense layer. The NN weights are quantized to 6-bit fixed-point data with 1 integer bit.

The ECON-Ts operate in a high-radiation environment due to their close proximity to particle collisions. 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 memory for longer than a cycle.

The ECON-T uses triple modular redundancy (TMR) to protect the NN weights against faults [18]. TMR is effective but incurs a 200% overhead [5, 40]. 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 is a tool that allows users to assess the sensitivity of NN weights to faults.[1] FKeras includes fast and efficient metrics that provide a bit-level sensitivity ranking of the weights. FKeras facilitates the addition of fault tolerance into the codesign problem by allowing the designer to consider how different quantized networks handle faults. FKeras provides a way to evaluate the fault tolerance alongside performance, resource usage, and power consumption, which can result in hardware accelerators that are smaller, more performant, and more resilient.

The primary goal of FKeras is to facilitate fault analysis with hls4ml [23]. hls4ml targets edge ML applications with low latencies, high throughput, minimal power budgets, and low resource usage [24]. FKeras extends the hls4ml workflow to assess the sensitivity of NN weights to faults, perform efficient fault injection (FI) campaigns, and facilitate design space exploration that considers fault tolerance alongside accuracy, performance, and resource usage.

hls4ml designs that target FPGAs must satisfy unique requirements compared to other architectures like GPUs and TPUs [29]. First, hls4ml implementations are highly quantized, often using

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

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 potentially modifies the sensitivity of computations to faults. QKeras [15] is a tool developed by Google and the hls4ml community to handle custom hardware data types in hls4ml. 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 most significant bits first, the gradient, and 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 bitwidths [19, 20, 42–44]. `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 [13], i.e., a more significant bit is at least as sensitive as a less significant bit, though there are exceptions. We can 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 the Hessian sensitivity metric to determine the most sensitive weights and flip those first. **??** shows that the Hessian sensitivity metric performs substantially better at guiding the FI towards faulty behaviors than randomly injecting faults as many FI tools and studies do [21, 32, 36].

We demonstrate the value of `FKeras` in considering fault analysis in the design space exploration process for a neural network. 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 [30] Our results show that different-sized networks have vastly different levels of fault tolerance, e.g., a smaller, less accurate network has more sensitive bits than a larger, more accurate network. Additionally, we show that the more significant bits are typically more sensitive than a less significant bit within a weight (confirming previous results [13]). This monotonicity generally holds within the same weight, but also across weights.

`FKeras` is a tool that helps design fault-tolerant neural networks in hls4ml. The primary contributions of `FKeras` are:

- Providing bit-level weight sensitivity metrics.
- Guiding fault injection campaigns to consider the most sensitive bits first.

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

2 FKERAS

NNs are often over-parameterized, and not all weights are equally important [26, 27], which indicates that some weights are more sensitive to faults than others. `FKeras` is a codesign tool for designing fault-tolerant neural networks 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.

2.1 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 [14, 33]. 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 [20, 44]. 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 [10, 20].

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 [44]. However, extracting the Hessian eigenvalues and eigenvectors takes

$O(n)$ memory in $O(n)$ time using techniques from randomized numerical linear algebra (RandNLA) [4, 22, 35, 41]. 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 [44]. Based on these k eigenvalues, we compute a parameter score:

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

where λ_i is the ith eigenvalue, v_i is the ith eigenvector, and p 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).

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 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 [44]. 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.

2.2 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 only expect faults to occur at most once per year. 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. We briefly describe the single-bit flip model in the remainder of this section. Information on the multi-bit flip model is available in [1].

2.2.1 Single-Bit Flip Model. A common fault model is the single-bit flip model [25, 32, 34], 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, or both. We limit our scope to only the weights of a NN, as motivated by the spatial dataflow architecture common to many edge NN hardware implementations.

3 EXPERIMENTAL EVALUATION

This section describes the experimental setup and results.

3.1 Experimental Setup

We demonstrate how FKeras can efficiently analyze a NN's fault sensitivity by performing experiments on CIFAR-10 [31] 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 [8], (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 [37]. 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.

ECON-T Medium is the model described in the paper by Di Guglielmo et al [18] - 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 [8]. 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-bit flip fault injection campaigns. 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

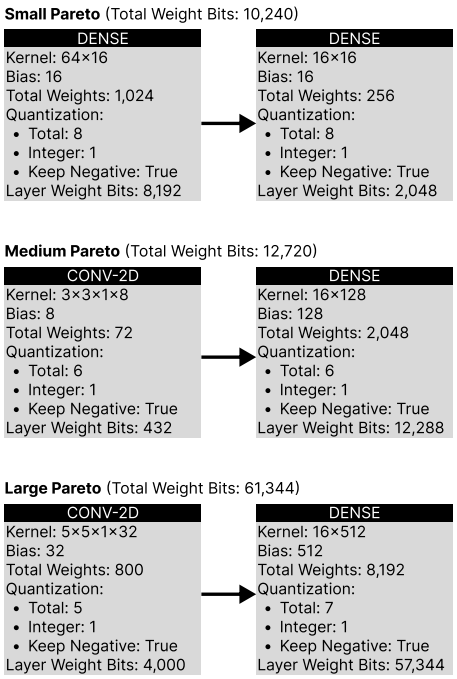


Figure 1: A layer-by-layer overview of the three ECON-T models.

inputs. We classify a bit as sensitive if flipping it causes the model error to exceed the average non-faulty model error.

3.2 Single Bit Flip Results

3.2.1 ECON-T Medium Model Resilience. We perform the first set of experiments on the ECON-T Medium Pareto autoencoder. Fig. 2 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 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 [13].

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 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.

3.2.2 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 [8]. 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 2. *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 2.1).

Next, we consider the relative magnitude of the error and not just the relative ranking. Fig. 3 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. 3a 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. 2 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 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. 3c)

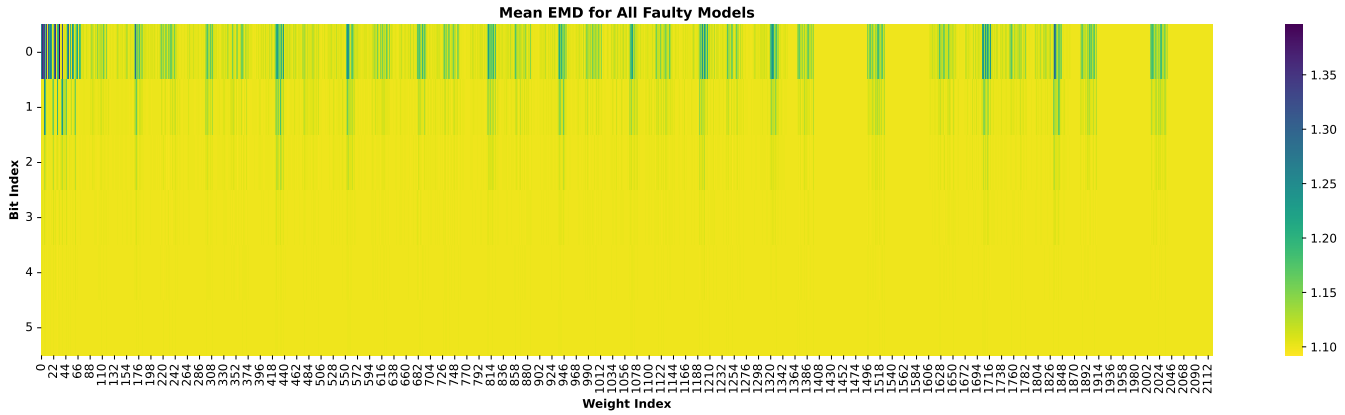
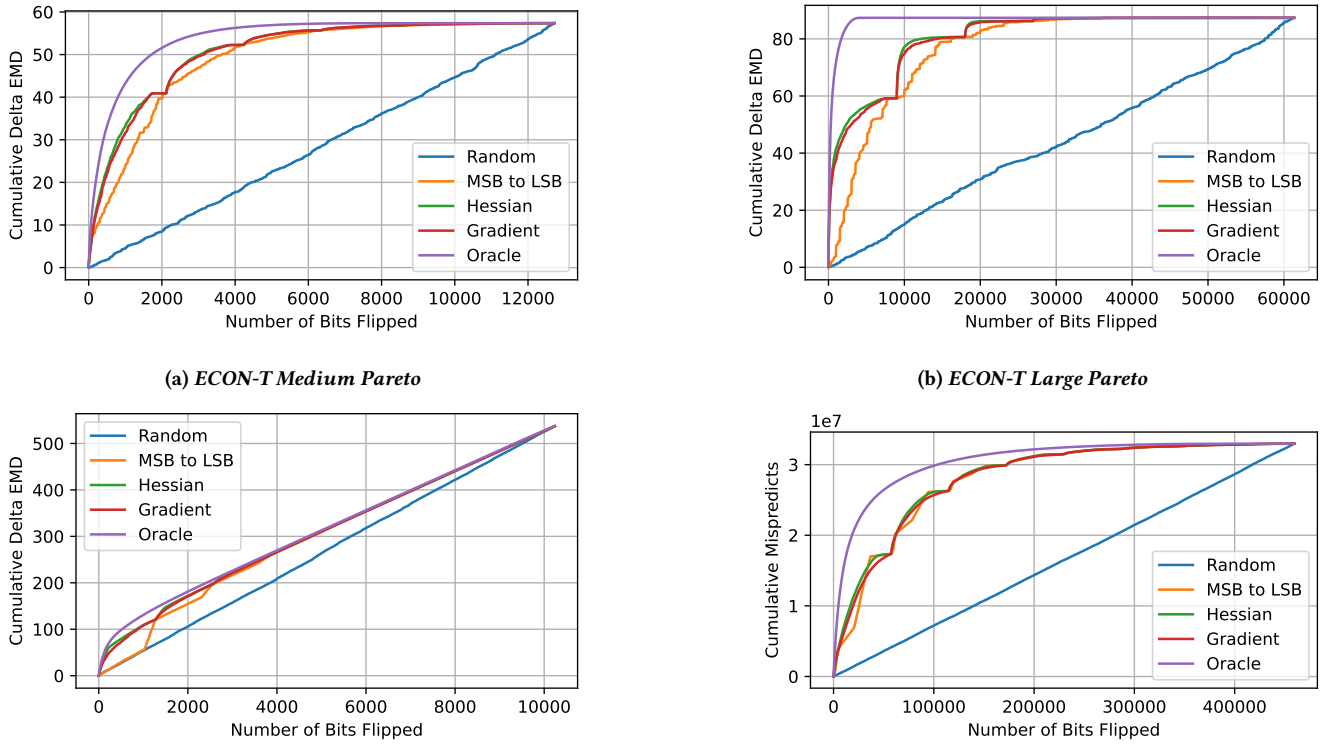


Figure 2: The average EMD for the ECON-T medium Pareto NN under a single-bit fault model. The x-axis corresponds to the NN weight. The y-axis represents the bit index of the weight where 0 is the MSB and 5 is the LSB.



(a) *ECON-T Medium Pareto*

(b) *ECON-T Large Pareto*

(c) *ECON-T Small Pareto*

(d) *CIFAR-10*

Figure 3: The magnitude of the error vs. the number of bits flipped for the four NN models. 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 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.

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. 3b) is the largest of the three. The large model has a small percentage of sensitive weights as indicated by the steep initial slop of the *Oracle*. In other words, the vast majority of the weights are insenstive 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.

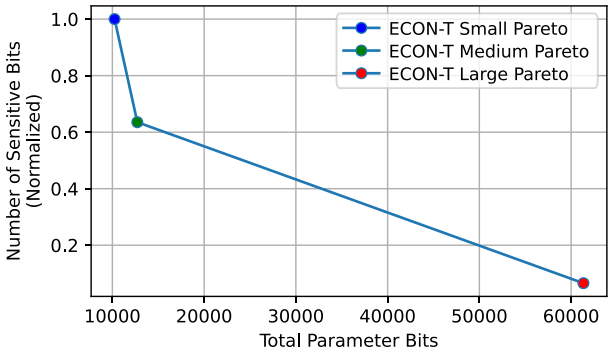


Figure 4: Part a) As model size increases, the percentage of sensitive bits in the model decreases. Part b) As *EMD* increases, the percentage of sensitive bits in the model increases.

Next, we summarize the relationship between model size and the sensitivity of its weights. Fig. 4 a) 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. 4 b) 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).

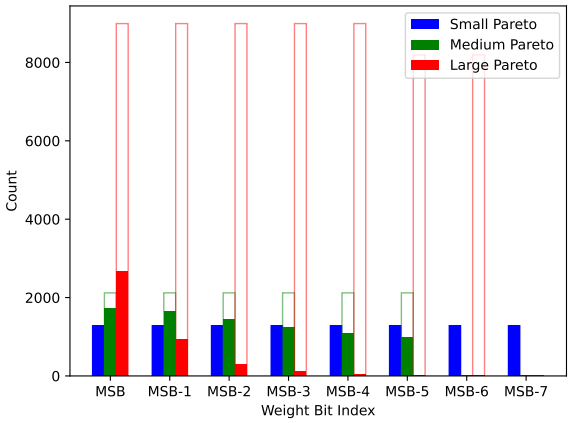


Figure 5: Distribution of the sensitive bits related to bit position for the three ECON-T NNs.

Fig. 5 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.

4 CONCLUSION

We develop `FKeras` as a tool to assess the fault tolerance of neural networks within the hls4ml framework. `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. In the models we studied, the most significant bits are generally the most sensitive and faults in the least significant bits generally induce little to no errors in the overall model. `FKeras` provided valuable insights for designing fault-tolerant, quantized hardware models with hls4ml.

REFERENCES

[1] 2023. FKeras. https://github.com/KastnerRG/fkeras.

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

[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] 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.

[5] 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

[6] 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).

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

[8] 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).

[9] 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.

[10] 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).

[11] 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.

[12] 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).

[13] 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.

[14] 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.

[15] 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]

[16] 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).

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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).

[25] 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.

[26] 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).

[27] 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).

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

[29] 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.

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

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

[32] 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.

[33] 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]

[34] 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.

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

[36] 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.

[37] 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

[38] 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.

[39] 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.

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

[41] 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.

[42] 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.

[43] 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.

[44] 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.