PrioriFI: More Informed Fault Injection for Edge Neural Networks

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

Nhan Tran
Fermi National Accelerator Laboratory
Batavia, IL, USA

Ryan Kastner
University of California San Diego
San Diego, CA, USA

Abstract

As neural networks (NNs) are increasingly used to provide edge intelligence, there is a growing need to make the edge devices that run them robust to faults. Edge devices must mitigate the resulting hardware failures while maintaining strict constraints on power, energy, latency, throughput, memory size, and computational resources. Edge NNs require fundamental changes in model architecture, e.g., quantization and fewer, smaller layers. PrioriFI is a more informed fault injection (FI) algorithm that evaluates edge NN robustness by ranking NN bits based on their fault sensitivity. PrioriFI prioritizes finding highly fault-sensitive bits, that is, the bits most critical to an NN's correctness, first. To accomplish this, PrioriFI uses the Hessian for the initial parameter ranking. Then, during an FI campaign, PrioriFI uses the information gained from past FIs as a heuristic so that future FIs target the bits likely to be the next most sensitive. With PrioriFI, designers can quickly evaluate different NNs and better co-design fault-tolerant edge NN systems.

CCS Concepts: • **Hardware** → **Hardware-software codesign**; **Transient errors and upsets**; • **Computing methodologies** → *Neural networks*.

Keywords: Edge AI; hardware-software codesign; neural network reliability; transient errors

ACM Reference Format:
Olivia Weng, Andres Meza, Nhan Tran, and Ryan Kastner. 2025. PrioriFI: More Informed Fault Injection for Edge Neural Networks. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '26)., To Appear, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

As machine learning (ML) becomes more capable, it is increasingly used in safety-critical applications, e.g., autonomous vehicles [4], healthcare [31], and scientific fields like high-energy physics and electron microscopy [12]. In these cases, ML algorithms must operate correctly when encountering soft errors [6]. Mitigating soft errors is important in applications that operate under harsh conditions [5, 12–14, 37].

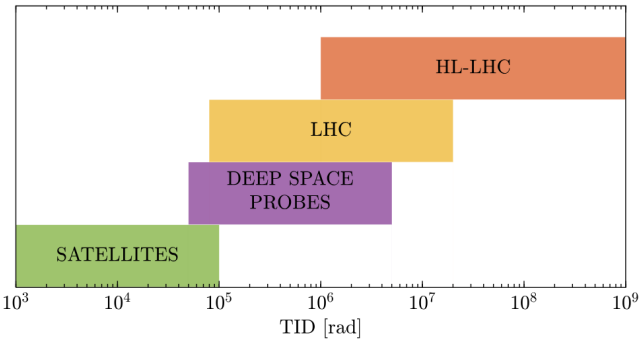


Figure 1. Edge NNs are deployed in high radiation environments. For example, high-energy physicists want to deploy edge NNs at the High Luminosity Large Hadron Collider (HL-LHC), which exhibits > 1000× the radiation in space when measured in Total Ionizing Dose (TID) [5]. Edge NNs deployed here will experience a fault every 15 seconds.

For example, high-energy physicists at the Large Hadron Collider (LHC) plan on deploying tens of thousands of end-cap concentrator (ECON-T) ASICs, each running a neural network (NN) encoder that must complete inference in 25 ns within an area budget of 4 mm^2 [13]. The ECON-T ASICs operate in the High Luminosity LHC (HL-LHC), which exhibits 1000× the radiation in space (see Fig. 1) [5]. A particle strike can cause a bit flip in NN weights, resulting in incorrect results. This edge NN must tolerate soft errors from particle strikes while operating within the given resource and latency constraints.

Edge NNs with tight latency and throughput constraints have unique characteristics, including heavy quantization and fully on-chip inference. Edge NNs often rely on custom hardware to meet such strict constraints and use design space exploration (DSE) to tune software and hardware to meet budgets on power, performance, and resources [7, 24, 28]. For instance, designers can use parity bits or triple modular

ASPLOS '26, Pittsburgh, PA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

redundancy (TMR) to protect their edge NNs—or no protection mechanisms at all if the edge NN is resilient enough for the given constraints. As a result, they need to understand how resilient their edge NN is to design the software and hardware accordingly. Previous research in NN fault tolerance often assumes that the weights and biases are stored in memory safeguarded by parity mechanisms [2, 9, 19], which is common for large NNs that retrieve weights and biases from off-chip memory [22]. Thus, previous work concentrates on faults in the outputs of NN layers, such as those in pipeline registers and arithmetic logic units. Yet, edge NNs often run on custom hardware, where designers choose how to protect their weights and biases, if at all. Since prior work has focused on large NNs under this fault model, the study of edge NN fault tolerance has been neglected [35].

Challenges. Most prior work evaluates NN fault tolerance by performing fault injection (FI) campaigns. FI campaigns simulate faults in software or hardware and then run multiple inputs through the NN to evaluate its performance. This approach is computationally intensive due to the need to simulate many faulty scenarios. Earlier work has aimed to reduce the scope of the search space, with a few that target edge NNs [9, 27]. BinFI [9] performs a binary search FI campaign that relies on *bit-level monotonicity*—that a more significant bit is at least as sensitive as a less significant bit—to reduce the search space exponentially. FKeras [35] uses the Hessian and bit-level monotonicity to sort the most significant bits (MSBs) of an NN's weights by its Hessian ranking first, followed by the same ranking on the second MSBs, and so on.

However, bit-level monotonicity is not a strict rule that holds for an NN's parameters. There are many exceptions where a less significant bit is more sensitive than a more significant bit—not only within a weight but also across the weights of a model. For the NNs we study in the paper, on average 15% of the NN bits violate bit-level monotonicity within a weight or bias and 80% violate monotonicity across weights or biases. These exceptions occur because faults get masked by operations that happen downstream in the NN, such as activation functions, leading to subpar FI campaign performance. BinFI suffers from falsely identifying weight or bias bits as sensitive or insensitive when bit-level monotonicity does not hold. FKeras injects faults in all of the MSBs before lesser significant bits, even though some lesser significant bits are more sensitive to faults than more significant bits. These exceptions in FKeras' Hessian ranking occur even for the least significant bits (LSBs) in the weights.

Contributions. This paper shows how to consider the exceptions in bit-level monotonicity during an FI campaign. *The key idea is to use the information gained during an FI campaign to dynamically prioritize the remaining campaign on the bits likely to be the next most sensitive, avoiding the pitfalls of assuming bit-level monotonicity.*

Our work PrioriFI[1] is a more informed FI algorithm that evaluates edge NN robustness by dynamically ranking NN bits based on their fault sensitivity. PrioriFI uses the Hessian for the initial weight and bias ranking. Then, during an FI campaign, PrioriFI leverages the information gathered from each FI to prioritize subsequent FIs on the bits most likely to exhibit high fault sensitivity. PrioriFI's goal is to guide FI campaigns to inject faults on the most sensitive weight and bias bits first because the sensitivity of these parameters is variable. Many do not result in faulty behavior. Thus, a campaign should focus on injecting faults into the bits most susceptible to faults. With PrioriFI, designers can quickly evaluate different NN architectures and co-design fault-tolerant edge NNs.

We evaluate the effectiveness of PrioriFI in incorporating fault analysis into edge NN DSE. We assess how PrioriFI performs on NNs trained on two high-energy physics datasets [14, 37] and the CIFAR-10 dataset [17]. While the scientific datasets represent applications in extreme, high fault-rate environments, the ML tasks themselves, e.g., classification, are common to many other non-scientific, edge applications [3]. By studying these datasets, we show that PrioriFI works well on scientific edge NNs and takes a step towards generalizing to more standard edge NNs similar to CIFAR-10. For each dataset, we study three NN architectures, namely a small, medium, and large NN, representing a spread from small, less accurate NNs to large, more accurate NNs. Our results show that PrioriFI avoids many exceptions to bit-level monotonicity, identifying more sensitive bits in FI campaigns sooner.

Our primary contributions are: (1) analyzing the shortcomings of relying on bit-level monotonicity to accelerate fault injection campaigns, (2) introducing PrioriFI[2] to dynamically prioritize fault injection campaigns on the most sensitive bits first, and (3) conducting NN architecture design space explorations that incorporate fault tolerance alongside traditional optimization criteria such as performance and resource utilization.

The paper is organized as follows. Sec. 2 discusses related work. Sec. 3 analyzes the pitfalls of bit-level monotonicity and introduces PrioriFI. Sec. 4 evaluates the fault tolerance of nine different edge NNs using PrioriFI. Sec. 5 concludes the paper.

2 Related Work

Identifying silent data corruptions (SDCs)—soft errors that lead to incorrect NN output [32]—is critical for fault tolerance. Three popular approaches are FI campaigns [21, 32], heuristics [10, 29, 30], and ML modeling [8, 33]. Few studies have focused on edge NNs that are quantized and operate

[1] PrioriFI is a portmanteau of "priority" and "FI", pronounced "priOR-uh-fy."
[2] PrioriFI is open-sourced at https://anonymous.4open.science/r/priorifi-submission.

Table 1. Comparing prior work to PrioriFI. If the work is just an FI tool, then we note its **No False Positives/Negatives** and **Dynamic FI** abilities as not applicable (–). Dynamic FI means this method uses the information it receives during an FI campaign to guide future FIs in the campaign.

FI Tool /Method	FI Speedup Method	No False Positives /Negatives	Dynamic FI
Ares [25]	✗	–	–
TensorFI [18]	✗	–	–
PyTorchFI [20]	✗	–	–
GoldenEye [23]	✗	–	–
enpheeph [11]	GPU support	–	–
[10]	Gradient	✓	✗
Aspis [29]	Gradient	✓	✗
StatFI [27]	Statistical sampling	✗	✗
FKeras [35]	Hessian-guided FI	✓	✗
BinFI [9]	Element-wise binary search	✗	✓
PrioriFI (this work)	Priority-guided FI	✓	✓

on specialized hardware, such as FPGAs and ASICs [34, 35]. Such hardware is critical for low-power sensing applications deployed at the edge. For the few studies that apply to edge NNs, we identify their limitations stemming from dependence on bit-level monotonicity. Tab. 1 compares PrioriFI to prior work.

FI campaigns simulate a fault either in software [9, 20, 23, 25] or hardware [15, 25] and pass a number of inputs through the NN to assess performance. FI campaigns are computationally intensive due to the vast number of faulty scenarios that must be simulated, particularly given the thousands to millions of parameters and operations involved in an NN [32]. Researchers have developed methods to speed up FI campaigns through heuristics, statistical FI, and more [9, 15, 21].

Prior work uses heuristics to determine NN sensitivity through measurements like the gradient [10, 29] or patterns in error propagation [30]. Aspis [29] uses a first-order Taylor expansion to rank NN weights by sensitivity. While inherently less accurate than FI campaigns, heuristics are cost-effective, sacrificing accuracy to achieve faster execution.

Statistical fault injection [27], which we call "StatFI", reduces the FI search space. StatFI relies on bit flips that lead to large swings in the weight's magnitude. 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 magnitude changes occur more frequently in floating point and less in fixed point, which are more common in edge NNs. Our results show that StatFI fails to identify many sensitive bits in edge NNs (as seen later in Fig. 7).

BinFI [9] relies on bit-level monotonicity, i.e., higher-order bits are at least as sensitive as lower-order bits, to reduce the number of faults to inject. BinFI performs a binary search within each layer output. BinFI starts by flipping the middle bit first, leading to two outcomes: (1) if it is sensitive, BinFI assumes all bits more significant than it are sensitive, continuing its binary search on the remaining lower order bits, or (2) if it is insensitive, BinFI continues its binary search on the bits more significant than it. BinFI is effective because it reduces the search space by $O(\log(n))$ per layer output, where n is the number of bits per output. Bit-level monotonicity generally works for edge NNs, so BinFI can find most of the sensitive bits in an edge NN. But, as previously mentioned in Sec. 1, faults do not always behave monotonically [9]. As we discuss later in Sec. 3.2, there are many exceptions to bit-level monotonicity—where a lower order bit is more sensitive than a higher order bit. In fact, our results show that for the nine NNs we study, on average 80% of the NN parameter bits violate bit-level monotonicity across weights and 15% of the bits violate bit-level monotonicity within a weight. Such violations lead BinFI to incorrectly identify bits as sensitive (false positives) or insensitive (false negatives), as seen later in Tab. 5.

FKeras [35] provides various metrics to rank weight bits by fault sensitivity, guiding FI campaigns to target the most sensitive bits first. FKeras' best sensitivity metric uses the Hessian[3]. FKeras' Hessian ranking speeds up FI campaigns but has subpar performance. Although the Hessian is effective at ranking the weights, it does not know their binary representation, so it cannot inform the relative sensitivity of the bits. FKeras thus relies on bit-level monotonicity to sort the individual bits of the weights. FKeras sorts all of the MSBs by Hessian weight ranking, followed by the second MSBs by the same weight ranking, and so on. But, there are many instances when a second MSB is more sensitive than a MSB—both within a weight and across weights (as seen in our results Fig. 2). These exceptions to bit-level monotonicity explain why FKeras' bit-level ranking is not as good as it could be.

In response to these shortcomings, PrioriFI provides a bit-level ranking that combines the Hessian with information learned during an FI campaign. The key idea is that information gained during an FI can guide the remainder of the campaign, avoiding the limitations of bit-level monotonicity. Our results show that PrioriFI identifies the most sensitive weight and bias bits with high accuracy and is thus a more effective FI algorithm (as seen later in Fig. 7).

3 PrioriFI

PrioriFI is a more informed fault injection algorithm for edge NNs. PrioriFI uses the Hessian and bit-level monotonicity to provide an initial parameter ranking. Then during an FI campaign, PrioriFI uses the sensitivity results it previously

[3]FKeras efficiently computes the Hessian eigenvalues and eigenvectors, taking $O(n)$ memory in $O(n)$ time, where n is the number of NN parameters, by using techniques from randomized numerical linear algebra [36].

collected to guide the campaign to avoid the exceptions to bit-level monotonicity. This section defines the fault model and our sensitivity metric. Then, we describe the limitations of assuming bit-level monotonicity in an NN. Finally, we provide algorithmic details of PrioriFI.

3.1 Fault Model

The single-bit flip model, which simulates scenarios where only one bit flips at a time, is widely used [9, 15, 21]. This fault model can be applied to NN weights, biases, activations, and even at finer granularity [16]. Since many edge NNs execute fully on-chip and require custom hardware, our study focuses on bit flips in NN weights and biases, as designers must consider how to protect them.

Given this fault model, we want to determine which NN weight and bias bits are sensitive to faults. Some bits are more sensitive than others, meaning flipping them results in significantly worse model performance than flipping others. Some bits are not sensitive, i.e., flipping them leads to correct results. Thus, we want to measure the different levels of fault sensitivity.

3.1.1 NN Sensitivity Metric. We define a sensitivity metric to measure which bits are more sensitive than others.

Definition 1 (Sensitivity Metric ΔC)**.**

$$\Delta C = \max(\overline{C}_{faulty} - \overline{C}_{faultless}, 0) \quad (1)$$

C is the cost of NN execution, $\overline{C}_{faulty}$ is the average cost with an FI, and $\overline{C}_{faultless}$ is the average cost without an FI. Thus, when flipped, a sensitive bit results in $\Delta C > 0$, and an insensitive bit results in $\Delta C = 0$, i.e., no cost to faulty NN execution.

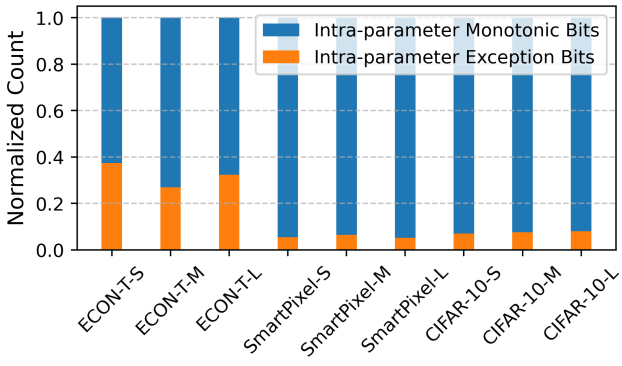
We determine C based on the NN's task at hand. For example, for classification tasks, C is the number of mispredicts. Given X, the correctly classified samples from the NN's validation dataset, we have $\forall x \in X, C_{faultless}(x) = 0$, so $\overline{C}_{faultless} = 0$. In other words, a sensitive bit leads to worse model performance when flipped.

3.1.2 Oracle. We perform an exhaustive FI campaign using the fault model and sensitivity metric to provide an *Oracle*. The Oracle knows the sensitivity metric value for every NN weight and bias bit when flipped. To generate the Oracle, we use FKeras's [35] FI capabilities to exhaustively perform single bit-flips for each NN (more details in Sec. 4.1). We use the Oracle to describe the limits of bit-level monotonicity (Sec. 3.2) and to evaluate the performance of PrioriFI (Sec. 4).

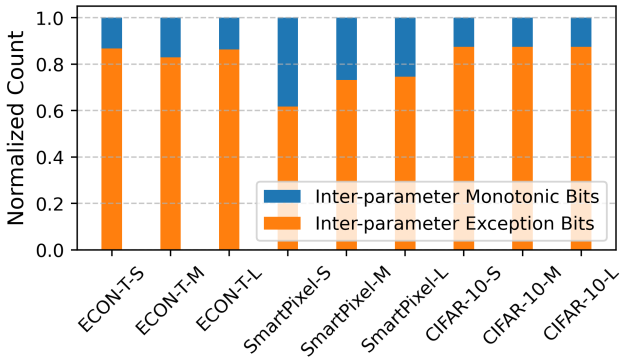
3.2 Limits of Bit-level Monotonicity

Bit-level monotonicity indicates that higher-order bits are more likely to be sensitive to faults than lower-order bits. Previous work uses bit-level monotonicity to accelerate FIs, e.g., BinFI [9] and FKeras [35]. But there are shortcomings.

Let us first define bit-level monotonicity more formally.



(a) Intra-parameter exceptions



(b) Inter-parameter exceptions

Figure 2. Distribution of intra- and inter-parameter exceptions of bit-level monotonicity. There are two ways to break bit-level monotonicity, where a lower-order bit exhibits higher sensitivity than a higher-order bit: (1) intra-parameter (within a parameter) and (2) inter-parameter (between parameters). In the nine NNs we study, on average 80% of a NN's parameter bits exhibit inter-parameter exceptions and 15% exhibit intra-parameter exceptions.

Definition 2 (Bit-level Monotonicity)**.** *Bit-level monotonicity says higher-order bits are at least as sensitive as lower-order bits:*

$$for\ x \geq y, \Delta C_x \geq \Delta C_y \quad (2)$$

where x is a higher-order bit and y is a lower-order bit, ΔC_x is the sensitivity of flipping bit x, and ΔC_y is the sensitivity of flipping bit y. This is based on the definition found in the BinFI paper [9].

Bit-level monotonicity is generally valid, especially within a parameter, because higher-order bits have a greater influence on the overall computation within an NN; however, this is not always true. Flipping higher-order bits causes a larger change in magnitude in an NN parameter compared with flipping a lower-order bit [9]. Prior work like StatFI [27] uses magnitude change to guide their FI campaign. Despite this intuition, there exist many instances when flipping a

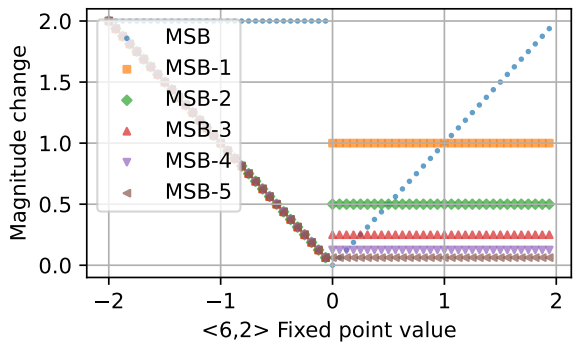


Figure 3. Applying ReLU to fixed-point values after a bit flip breaks bit-level monotonicity. In this example, we flip a bit for every value in the $< 6, 2 >$ fixed-point representation (6 bits total, 2 bits integer) and apply a ReLU activation function to the result, plotting the resulting change in magnitude. Note that the brown-dotted diagonal on the left represents the overlapping of all the lower-order bits (MSB-1 and lower). Without ReLU, bit flips are monotonic. With ReLU, bit flips are not strictly monotonic, as seen in prior work [9, 21].

higher-order bit leads to a lower ΔC compared with flipping a lower-order bit, violating bit-level monotonicity [9, 21].

In fact, there are two types of exceptions to bit-level monotonicity: (1) *intra-parameter* (within a parameter in Fig. 2a) and (2) *inter-parameter* (between parameters in Fig. 2b). Based on Def. 2, an NN parameter bit violates *intra-parameter* monotonicity if it is less sensitive than a lower-order bit in the same parameter. A bit violates *inter-parameter* monotonicity if it is less sensitive than a lower-order bit found in any of the other parameters. We count the number of bit-level monotonic exceptions in the nine edge models we study later in Sec. 4, plotting them in Fig. 2. These models are trained on three edge datasets: the HGCal dataset [14], the Smart Pixel dataset [37], and CIFAR-10 [17]. For each dataset, we trained a small, medium, and large model to represent a spread in model size, accuracy, and fault sensitivity. Note, the ECON-T model is trained on the HGCal dataset. More details are found in Sec. 4.1. As seen in Fig. 2, there are significantly more inter-parameter than intra-parameter exceptions. On average, 80% of the bits exhibit inter-parameter exceptions and 15% exhibit intra-parameter exceptions. This is likely due to the fact that there are many more lower-order bits across parameters compared to within a single parameter. These exceptions can occur because faults are masked by operations that happen downstream in the NN, such as activation functions, leading to non-monotonic behavior.

In Fig. 3, we present how masking causes non-monotonic behavior in edge NNs. In this example, we select a $< 6, 2 >$ fixed-point representation, where there are 6 total bits and

Algorithm 1: PrioriFI

```
let N = NN model
let p = NN parameters
let p_ranked = hessianRanking(p)
let Bs = sortBitsByHessian(p_ranked)
Note: Bs is a list of ranked bit lists sorted by bit
  significance order, i.e., [[MSBs], [MSB-1s], ...]
let deltaCs = {} // store ΔCs here
let k = how many of the last ΔCs to consider
Function nextBitFlip(bitLists, deltaCs, k):
    ms = {}
    for i in len(bitLists) do
        m = medianLastK(deltaCs[i], k)
        ms[i].append(m)
    end
    a = argsort(ms, reverse=True)
    for i in len(a) do
        if bitLists[i] is not empty then
            nextBit = a[i]
            break
        end
    end
    return nextBit
// Flip first bit in each bit list
for i in len(Bs) do
    N' = flipBit(Bs[i][0], N)
    d = deltaC(N') // Measure ΔC of N'
    deltaCs[i].append(d)
    Bs[i].pop()
end
// Priority-based FI
while ∀b ∈ Bs, b is not empty do
    f = nextBitFlip (Bs, deltaCs, k)
    N' = flipBit(Bs[f][0], N)
    d = deltaC(N')
    deltaCs[i].append(d)
    Bs[f].pop()
end
```

2 integer bits (one of which is a sign bit). This is the data type that the ECON-T-M uses to represent its parameters; however, the trends presented here generalize to other signed fixed-point representations. For each $< 6, 2 >$ value, we flip a bit, one of MSB, MSB-1 (second MSB), etc., and then apply ReLU (Fig. 3), a common activation function used in NNs. We then plot the resulting magnitude change for that value.[4] With the ReLU activation function, which itself is

[4] In Fig. 3, the brown-dotted diagonal on the left represents the overlapping of all the lower-order bits (MSB-1 and lower).

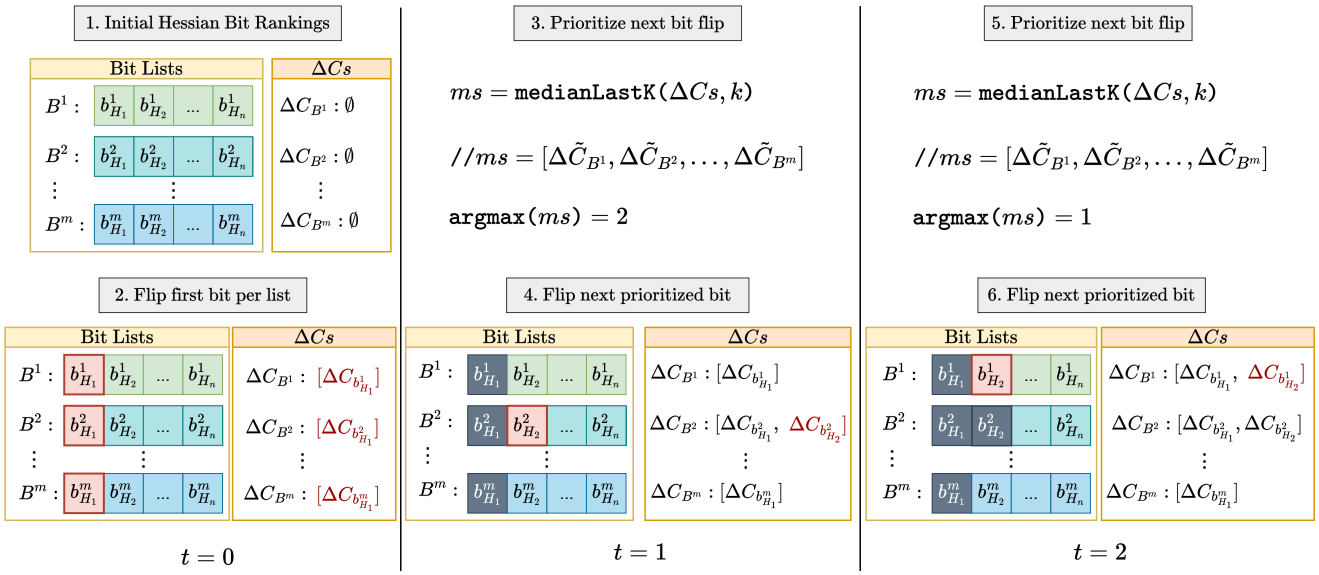


Figure 4. The PrioriFI algorithm in action. For an NN with n weights and biases quantized to m bits, we first rank the bits of the same significance using the Hessian from most to least sensitive. For instance, the MSBs are in list B^1, where the highest ranked MSB is $b^1_{H_1}$, where H_1 is the highest Hessian ranking. B^2 contains the second MSBs and so on till we reach B^m the LSBs. Then, we flip the first bit in each of the bit lists, i.e., the highest Hessian-ranked bits, and record the corresponding ΔCs. This ends the initial phase ($t = 0$). Next, we enter the priority-based FI phase. For the first iteration ($t = 1$), we take the median of the last k measurements for each ΔC_{B^i}, resulting in a list of medians ms, which we denote as $\Delta\tilde{C}_{B^i}$. Note the ms are sorted in bit significance order, where $\Delta\tilde{C}_{B^1}$ is first, then $\Delta\tilde{C}_{B^2}$, and so on till $\Delta\tilde{C}_{B^m}$. Then we take the argmax of the ms, i.e., $\max_i \Delta\tilde{C}_{B^i}$. Note, this is a simplified version of the `nextBitFlip()` function in Alg. 1, which ensures the argmax corresponds to the highest-ranked non-empty B^i list. In the $t = 1$ case, the argmax is 2, so we flip the next bit in the B^2 list and record its ΔC. This process continues in the next iteration ($t = 2$).

monotonic, magnitude changes behave non-monotonically—higher-order bits may result in smaller magnitude changes compared to lower-order bits, e.g., fixed point value 0.5. We thus observe non-monotonic behavior within a fixed-point value due to masking by a ReLU.

Thus, there exist many exceptions to bit-level monotonicity, especially across parameters (inter-parameter). These exceptions can occur because faults are masked in a variety of ways as they propagate through the NN, as seen in Fig. 3.

3.3 PrioriFI Algorithm

Although bit-level monotonicity provides a first-order approximation for bit sensitivity, it has numerous exceptions, particularly across different parameters. Thus, PrioriFI uses bit-level monotonicity as a starting point and then uses methods that consider *inter-parameter* importance to avoid the many exceptions to monotonicity, especially given how prevalent inter-parameter exceptions are (see Fig. 2b). During an FI campaign, PrioriFI uses information gained on the fly to guide its next FI to prioritize inter-parameter exceptions. This way, it avoids the pitfalls of bit-level monotonicity, targeting the most sensitive bits first.

Alg. 1 describes the PrioriFI algorithm in detail. PrioriFI begins by ranking the bits of each significance, e.g., MSB, MSB-1, etc., using FKeras' Hessian sensitivity metric. This metric has been shown to be effective at ranking NN parameters by fault sensitivity [35]. In the initial phase, PrioriFI flips the highest-ranked bit of each significance, measuring the ΔC per bit flip. Next, in the priority-based FI phase, PrioriFI takes the median of the last k ΔC measurements per bit significance. The highest median corresponds to the bit significance that the FI campaign should flip next, e.g., the next highest-ranked MSB-1. By using the ΔC measurements to guide our FI campaign, we account for inter-parameter exceptions to bit-level monotonicity, prioritizing these exceptions when they occur to find the more sensitive bits earlier. Fig. 4 demonstrates the first few iterations of the PrioriFI algorithm, providing a detailed explanation.

4 Evaluation

In this section, we describe our experimental setup and evaluate PrioriFI.

4.1 Experimental Setup

We demonstrate how PrioriFI efficiently performs an FI campaign on three edge datasets: the HGCal dataset [14], the Smart Pixel dataset [37], and CIFAR-10 [17]. The HGCal and Smart Pixel datasets are two high-energy datasets. Both of these datasets represent edge applications that occur in the HL-LHC, a high-radiation environment, as previously shown in Fig. 1. While these datasets represent applications in extreme fault-rate environments, the ML tasks themselves are common to many other non-scientific, edge applications [3].

The HGCal dataset contains sensor images from high-energy particle collision sensor data. The ECON-T autoencoder previously mentioned (Sec. 1) is trained on the HGCal dataset. The goal of the ECON-T is to compress this data to a lower-dimension latent space (and then later decode it offline for further processing) [13]. We only evaluate the fault tolerance of the encoder (and not the decoder) because it is the only part of the NN exposed to radiation. The Smart Pixel dataset contains vectors representing 13 key features of high-energy particle clusters from sensor data. The goal of an edge NN trained on Smart Pixel data is to classify the clusters into three classes of high-momentum and low-momentum particles. CIFAR-10 is a popular image classification dataset.

For each dataset, we study three differently sized edge NN architectures, in particular, a small, medium, and large NN. Each model is quantized to a low-bitwidth (ranging from 3 to 8 bits) fixed-point representation due to resource constraints at the edge. This represents a spread of models that researchers often consider when they perform design space exploration to find an optimal edge model for their given constraints, such as model performance, latency, and resources [7, 24, 28]. In our experiments, the three models per dataset represent a tradeoff between model accuracy and size. The ECON-T autoencoders feature convolutional and dense layers, the SmartPixel NNs are fully-connected, and the CIFAR-10-S and -M NNs are convolutional whereas CIFAR-10-L is a ResNet8 [14]. Fig. 10 in the appendix provides a detailed description of the NN architectures.

We use FKeras [35] to perform our FI campaigns on the weights and biases of each model. We ran our experiments on an AMD Ryzen Threadripper PRO 7985WX 64-Cores CPU and an NVIDIA RTX 4500 Ada Generation GPU. We first generate the Oracles for each model. For HGCal, we run each FI on 20 000 validation inputs. We evaluate ECON-T NN sensitivity using a ΔC where C is Earth Mover's Distance (EMD) [26]. EMD is the reconstruction loss between the input and the decoded output, measuring the distance between two probability distributions. An EMD of 0 indicates a lossless reconstruction. Since our ECON-T NNs are lossy, we define $\overline{C}_{faultless} = \overline{\text{EMD}}$, i.e., the average EMD of the model under non-faulty conditions. For Smart Pixel and CIFAR-10, we run each FI on a subset of the 11 113 and 10 000 validation inputs, respectively, namely the inputs that each model classifies correctly. We evaluate Smart Pixel and CIFAR-10 NN sensitivity using ΔMispredicts. Tab. 2 describes the baseline model performance, the total number of weight and bias bits, and the number of FI validation samples for each edge NN.

Table 2. Experimental setup. For CIFAR-10 and SmartPixel, the **# FI samples** is the number of correctly classified samples out of 10 000 and 11 113 total validation samples, respectively.

NN	EMD/ Acc. (%)	Total weight & bias bits	# of FI samples
ECON-T-S	1.55	10 496	20 000
ECON-T-M	1.10	12 864	20 000
ECON-T-L	0.81	61 616	20 000
SmartPixel-S	70.5	825	7 835
SmartPixel-M	72.9	3 956	8 098
SmartPixel-L	74.2	34 828	8 244
CIFAR-10-S	77.3	319 056	7 727
CIFAR-10-M	83.1	460 656	8 313
CIFAR-10-L	86.2	944 208	8 616

4.2 More informed fault injection

We evaluate the efficiency of PrioriFI with respect to its ability to guide an FI campaign. Our evaluation is conducted on the nine edge NNs (Tab. 2). We compare PrioriFI's FI performance against the following FI algorithms:

1. *Random* produces a ranking where are all bits are included in a random order.
2. *MSB-to-LSB* produces a ranking where are all bits are ordered according to their bit significance such that all MSBs come before all MSB-1s, etc.
3. *BinFI* produces a ranking based on a binary FI algorithm [9]. BinFI was originally designed to target the activations, but we adapt it to target the weights and biases. This approach has three variants that are described later in this section.
4. *StatFI* produces a ranking based on a statistical FI algorithm [27]. This approach has data-aware and data-unaware variants that adjust the sample size used in their algorithm.
5. *Taylor* as presented in Aspis [29] introduced a ranking based on the first-order Taylor expansion to sort the weights and biases by importance. Since they do not provide a bit-level ranking, we apply an MSB-to-LSB ordering to its initial ranking.
6. *Hessian* produces a bit-level ranking by ordering bits based on a parameter-level Hessian metric and then applying an MSB-to-LSB ordering from there. This ranking was introduced in [35].

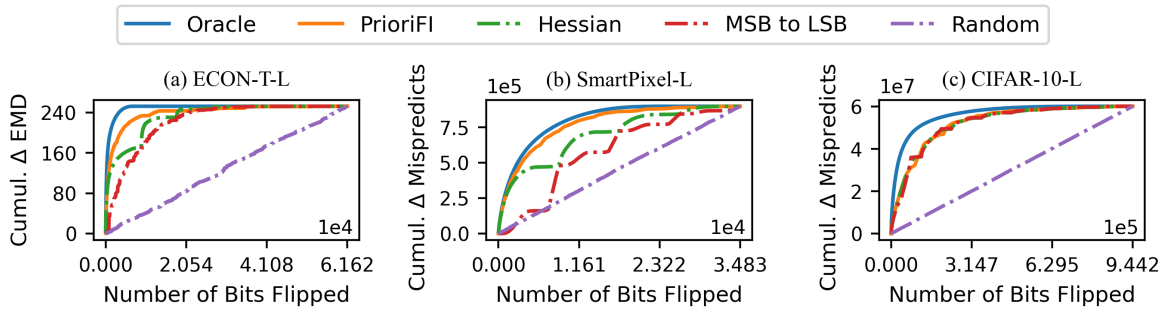


Figure 5. PrioriFI improves on rankings relying on bit-level monotonicity. We measure the cumulative ΔC as we flip more bits in each NN. Each model plots five rankings that attempt to order the NN weight and bias bits from those that contribute the most error to those that contribute little or no error. We see that PrioriFI adjusts for the exceptions to bit-level monotonicity found in the MSB-to-LSB and Hessian rankings, finding more sensitive bits faster. Note that the MSB-to-LSB ranking has few exceptions to bit-level monotonicity for CIFAR-10, so there is less room for improvement with PrioriFI.

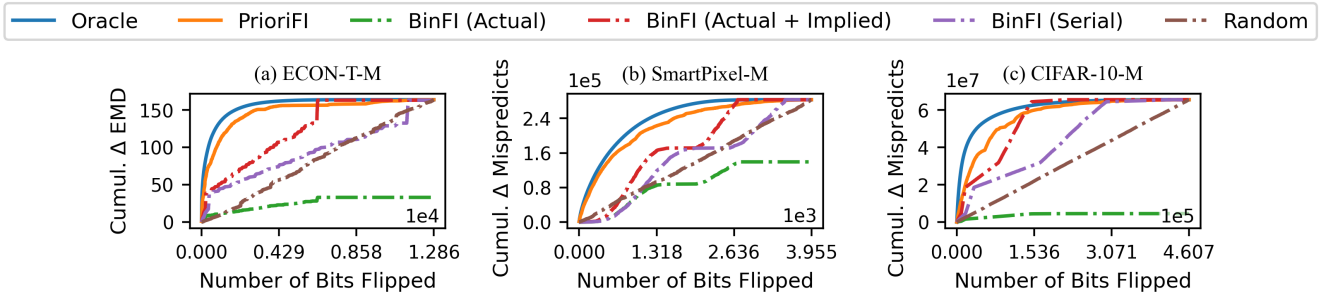


Figure 6. The spread of BinFI rankings. BinFI reduces the number of FIs in a campaign by implying bits to be sensitive. *BinFI (Actual)* are the actual bits that BinFI flips, representing the cumulative ΔC it actually learns. *BinFI (Actual + Implied)* includes the actual and implied sensitive bits that BinFI learns, representing the cumulative ΔC it theoretically could attain. *BinFI (Serial)* is derived from flipping the *BinFI (Actual + Implied)* bits in order, representing the cumulative ΔC it truly learns if it were to flip the implied bits too. *BinFI (Actual)* falls below the *Random* ranking because it does not flip all bits in the model.

7. *Oracle* produces a perfect ranking of the bits as described in Sec. 3.1.2.

Based on our Oracles, we found a different percentage of sensitive bits per NN per dataset. For the ECON-T NNs, 100%, 82%, and 11% of the bits were sensitive for ECON-T-S, ECON-T-M, and ECON-T-L, respectively. For the SmartPixel NNs, 86%, 79%, and 64% of the bits were sensitive for SmartPixel-S, SmartPixel-M, and SmartPixel-L, respectively. For the CIFAR-10 NNs, 83%, 87%, and 76% of the bits were sensitive for CIFAR-10-S, CIFAR-10-M, and CIFAR-10-L, respectively.

Fig. 5 features the results from our first set of experiments, demonstrating how PrioriFI improves upon rankings that rely on bit-level monotonicity. In these experiments, we compare the performance of PrioriFI to that of the Hessian and MSB-to-LSB which base their rankings on bit-level monotonicity. The Oracle ranking and Random ranking provide the best and worst bounds, respectively, on the performance one can expect from an FI campaign on the edge NNs under analysis. In Fig. 5a and Fig. 5b, the Hessian outperforms MSB-to-LSB, and PrioriFI outperforms the Hessian. In this context, outperforming equates to requiring fewer FIs (bit flips) to identify the model bits that produce the greatest ΔC. The plateaus in the Hessian ranking appear because it fails to account for exceptions to bit-level monotonicity. PrioriFI has a smooth curve without plateaus because it dynamically adjusts for these exceptions.

Fig. 6 showcases the results from our experiments exploring the performance differences between three variants of BinFI. *BinFI (Actual)* represents the ΔC that is actually measured by flipping a bit when conducting an FI campaign guided by BinFI. *BinFI (Actual + Implied)* represents the theoretical maximum performance of a BinFI approach when multiple bits are implied as being sensitive based on a single bit flip in a lower-order bit. However, the ΔC of the implied bits cannot be verified without actually flipping a bit, so the theoretical gains in information are not visible to the FI process—they must be assumed. *BinFI (Serial)* represents a slightly less efficient (with respect to the number of bit

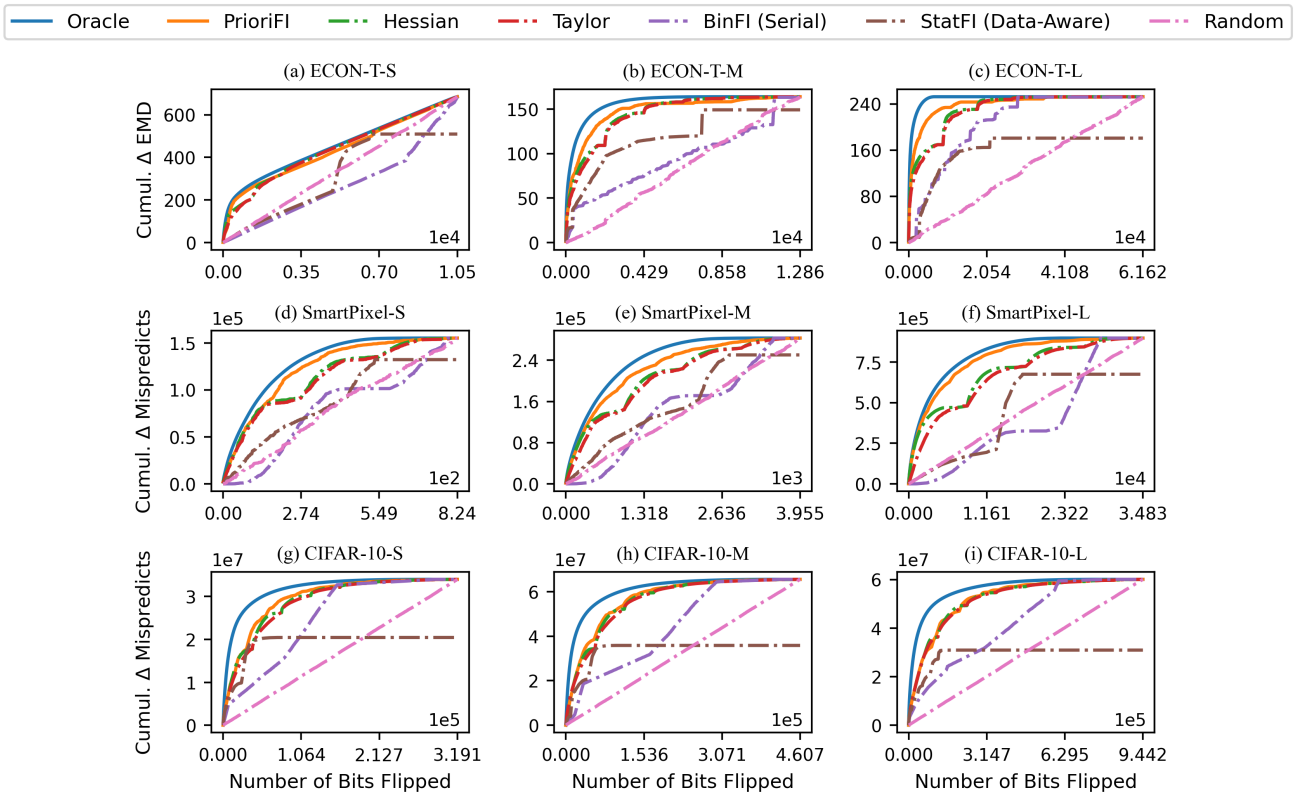


Figure 7. Comparing PrioriFI to related work. PrioriFI finds more sensitive bits faster than related work does. It accumulates more ΔC error in fewer bit flips, i.e., identifying the more critical sensitive bits faster.

flips) approach that aims to remove ambiguity in the implied bits by flipping them to confirm they are indeed sensitive bits. We adopt the serial variant of BinFI in the remainder of our experiments since it fits best with the ambiguity/risk tolerance of the target applications for our NNs.

Fig. 7 shows PrioriFI's advantage over FI approaches from the related work. PrioriFI consistently outperforms the other approaches. Across a variety of NN architectures (the rows in Fig. 7) and model sizes (the columns in Fig. 7), PrioriFI's dynamic approach yields greater cumulative ΔC earlier in the FI process, as evidenced by the smoothness of its curves. The "bumps" that occur in the related work demonstrate how they fail to account for non-monotonic behavior. We further quantify these results by computing the area under the curve (AUC) for each of the rankings shown in Fig. 7 and dividing it by the oracle's AUC, as shown in Tab. 3. We use AUC because we want to quantify how close to the oracle the other rankings are in this chart. Since the oracle is a perfect ranking, we present its AUC as 1. PrioriFI is the closest to the oracle ranking with the highest AUCs for seven out of the nine models evaluated. Even for ECON-T-S, PrioriFI is only 0.01 AUC worse than the Hessian.

In Tab. 3, PrioriFI improves on prior work such as the Hessian. The Hessian performs well in cases where bit-level monotonicity generally holds, but even in these cases, PrioriFI works nearly as well (ECON-T-S), if not the same (CIFAR-10-L). PrioriFI relies on the quality of the initial Hessian ranking and how often inter-parameter exceptions to bit-level monotonicity occur. For the ECON-T and SmartPixel models, the Hessian provides a good ranking, as seen in how close the Hessian ranking is to the oracle ranking in the first few bit flips in Fig. 7. But, this trend is seen less in the CIFAR-10 models, as the Hessian only provides an approximation of sensitivity. We also see in Fig. 5c that CIFAR-10-L does not have as many bit-level monotonic exceptions, since the MSB-to-LSB ranking already works quite well. Even so, PrioriFI improves on the rankings relying on MSB-to-LSB monotonicity, like the Taylor and Hessian. In contrast, for the ECON-T and SmartPixel models, we observe the poor performance of the MSB-to-LSB ranking as well as the many plateaus in the Hessian ranking, indicating that there are many exceptions to bit-level monotonicity.

To better understand why the CIFAR-10 NNs do not have many monotonic exceptions, we observe that not all exceptions are equal. Although there are many inter-parameter exceptions (previously seen in Fig. 2b), we can imagine that a given higher-order bit is less sensitive than only a few lower-order bits in the NN, whereas another higher-order

Table 3. Quality of PrioriFI's ranking using AUC. We normalize each ranking's area under the curve (AUC) from Fig. 7 to the oracle's AUC to quantify how quickly each ranking finds sensitive bits. PrioriFI improves on seven out of the nine models.

Ranking	ECON-T			SmartPixel			CIFAR-10		
	S	M	L	S	M	L	S	M	L
Oracle	1	1	1	1	1	1	1	1	1
Random	0.75	0.51	0.53	0.59	0.59	0.56	0.54	0.54	0.54
StatFI (data-aware)	0.70	0.76	0.64	0.69	0.66	0.57	0.61	0.55	0.52
BinFI (Serial)	0.60	0.61	0.83	0.61	0.61	0.50	0.79	0.73	0.75
Taylor	0.96	0.92	0.92	0.89	0.86	0.84	0.91	0.92	0.92
Hessian	**0.97**	0.93	0.93	0.88	0.88	0.87	0.92	0.93	**0.93**
PrioriFI	0.96	**0.95**	**0.96**	**0.95**	**0.94**	**0.97**	**0.93**	**0.94**	**0.93**

bit is less sensitive than many more lower-order bits. Thus, some parameter bits have more severe inter-parameter monotonic exceptions than others. To quantify the severity of a bit's monotonic exceptions, or lack thereof, we develop a metric called *monoscore*. It measures how monotonic or non-monotonic a bit is at an inter-parameter level. More formally, the monoscore is defined below.

Definition 3 (Monoscore)**.** *Given a NN with n bits, the monoscore of a bit is defined as:*

$$monoscore(b) = \frac{1}{n}\sum_{i=0}^{n} score(b, d_i) \tag{3}$$

where b is a given bit and d_i is the i-th bit. $score(\cdot)$ is defined as:

$$score(b^x, d^y) = \begin{cases} +1 & x > y, \Delta C_{b^x} \geq \Delta C_{d^y} \quad (4a) \\ +1 & x < y, \Delta C_{b^x} \leq \Delta C_{d^y} \quad (4b) \\ 0 & x = y \quad (4c) \\ -1 & x > y, \Delta C_{b^x} < \Delta C_{d^y} \quad (4d) \\ -1 & x < y, \Delta C_{b^x} > \Delta C_{d^y} \quad (4e) \end{cases}$$

where b^x is a bit with significance x, d^y is a bit with significance y, Eq. 4a and Eq. 4b define monotonic behavior, Eq. 4d and Eq. 4e define non-monotonic behavior, and Eq. 4c defines when bits b and d have equal significance, e.g., both are MSBs and are neither monotonic nor non-monotonic.

In other words, to compute the monoscore for a given bit, we first compare it with another bit in the NN. We add +1 if it behaves monotonically with the other bit, 0 if the two bits have the same bit significance, and -1 if it behaves non-monotonically with the other bit. We do this for all the bits in the NN and divide by the total number of NN bits to get an average for how monotonic this bit is relative to the rest of the NN bits. If a bit exhibits strictly monotonic behavior, its monoscore will be 1, and if it exhibits strictly non-monotonic behavior, its monoscore will be -1. A monoscore of 0 means the bit behaves monotonically with half the bits in the NN and non-monotonically with the other half.

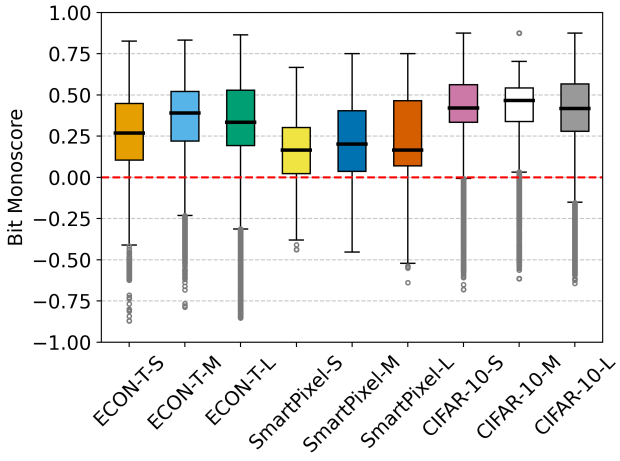


Figure 8. Distribution of the bit monoscores per model. None of the NNs have bits that are strictly monotonic (monoscore = 1) or strictly non-monotonic (monoscore = -1). The SmartPixel NNs have the lowest median monoscores, meaning they are the most non-monotonic compared with the other models, whereas the CIFAR-10 NNs have the highest median monoscores, indicating they are the most monotonic. PrioriFI is thus better at finding the sensitive bits in the SmartPixel NNs compared with the CIFAR-10 NNs.

To compute the monoscore for a whole model, or the *model monoscore*, we average all the bit monoscores of an NN's bits like so:

$$\text{model monoscore} = \frac{1}{n}\sum_{i=0}^{n} \text{monoscore}(b_i) \tag{5}$$

where n is the total number of NN bits. Similarly to the bit monoscore, a perfectly monotonic NN has a model monoscore of 1, a perfectly non-monotonic NN has a score of -1, and a NN that is equally monotonic and non-monotonic has a score of 0.

Given the monoscore defined above, we now analyze how monotonic the nine edge NNs are. We compute the

Table 4. Model monoscores. The CIFAR-10 NNs are the most monotonic, and the SmartPixel NNs are the least monotonic, so PrioriFI is more effective on the SmartPixel NNs than the CIFAR-10 NNs.

Model	Model monoscore
ECON-T-S	0.26
ECON-T-M	0.35
ECON-T-L	0.32
SmartPixel-S	0.16
SmartPixel-M	0.22
SmartPixel-L	0.25
CIFAR-10-S	0.44
CIFAR-10-M	0.43
CIFAR-10-L	0.40

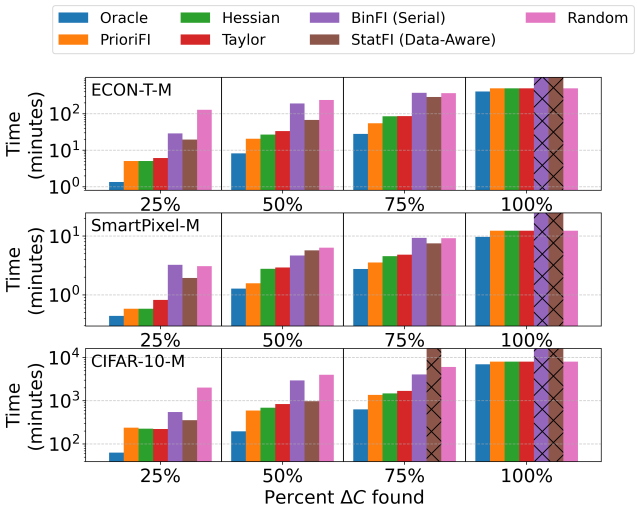


Figure 9. PrioriFI time savings. PrioriFI saves FI time, finding the bits contributing to 25/50/75% of the NN's sensitivity in less or comparable time compared to prior work. When the bar reaches the top of the y-axis and contains hatch marks, it means this strategy never finds that percentage of sensitive bits due to identifying false negatives.

monoscore for every bit in each edge NN and plot its distribution in Fig. 8. In this chart, we see that none of the NNs are perfectly monotonic or non-monotonic. The SmartPixel NNs have the lowest median monoscores and lowest model monoscores (Tab. 4), suggesting that they are the most non-monotonic compared with the other NNs. As such, PrioriFI significantly improves the SmartPixel FI campaigns compared to prior work (Tab. 3), successfully adjusting for the exceptions to bit-level monotonicity, as seen in Fig. 7. The CIFAR-10 NNs have the highest median monoscores and the highest model monoscores (Tab. 4), so they are the most monotonic compared to the other NNs. As a result, PrioriFI is unable to significantly improve the CIFAR-10 FI campaigns because there is not a lot of non-montonic behavior to compensate for (Fig. 7). PrioriFI's performance on the ECON-T NNs splits the difference between SmartPixel and CIFAR-10 NNs, which corresponds to how the ECON-T monoscores (Fig. 8 and Tab. 4) similarly split the difference. Whether a model violates bit-level monotonicity a lot or not depends on each individual model's idiosyncrasies, which is hard to predict. Nevertheless, PrioriFI shines when there are many exceptions to bit-level monotonicity, especially at the inter-parameter level.

We quantify the FI time saved using PrioriFI in Fig. 9. In this study, we measure how much time it takes for each ranking to find the bits that contribute to 25%, 50%, 75%, and 100% of the total cumulative ΔC in the medium-sized models. PrioriFI saves FI time because it can find the most sensitive bits in less time than prior work. For instance, PrioriFI finds 50% of SmartPixel-M's cumulative ΔMispredicts 43% faster than the Hessian. In another instance, PrioriFI finds 50% of CIFAR-10-M's ΔMispredicts 14% faster than the Hessian, saving one hour and 37 minutes of time spent injecting faults. But all techniques struggle to find the "tails," spending much more time after finding 75% of the sensitive bits to find the remaining ones. Since StatFI and BinFI are unable to identify all of the sensitive bits, their bars sometimes reach the top of the chart and contain hatch marks, meaning they can never find the bits contributing a given percentage of the ΔC. This shows how StatFI and BinFI suffer from false negatives.

BinFI and StatFI suffer from incorrectly identifying bits as sensitive (false positives) or insensitive (false negatives) as seen in Tab. 5. StatFI suffers from these errors because it computes a subset of all NN model bits to sample in its FI campaign. This sample size calculation is based on large changes in magnitude. However, as shown in Sec. 3.2, magnitude changes from bit flips can be masked by operations that occur downstream in an NN. Moreover, these large changes in magnitude occur more often in floating point, which has a large range in the values it represents. Because fixed-point datatypes have such a small range, large swings in magnitude do not occur. Thus, *StatFI (data-aware)*'s reliance on magnitude change to determine its statistical FI sampling is flawed for the quantized representation present in edge NNs. BinFI encounters these false positive/negative errors because it relies on bit-level monotonicity to infer the sensitivity of bits. Although the rates are low, missing these amounts of sensitive bits can have a significant impact on resource constraints on edge NNs when considering how many resources can be dedicated to protecting the sensitive bits. PrioriFI flips all bits from high sensitivity to low, so it does not suffer from false positives or false negatives.

Table 5. False Positives and False Negatives in FI algorithms. BinFI and StatFI suffer from incorrectly identifying bits as sensitive (false positives) or insensitive (false negatives). PrioriFI does not suffer from false positives or negatives. (A + I) = Actual + Implied. (D-A) = Data-aware.

NN	FI Method	# False Positives (%)	# False Negatives (%)
ECON-T-M	BinFI (A + I)	210 (1.6%)	669 (5.2%)
	StatFI (D-A)	889 (6.9%)	4 021 (31.3%)
	PrioriFI	**0 (0%)**	**0 (0%)**
SmartPixel-M	BinFI (A + I)	28 (0.7%)	28 (0.7%)
	StatFI (D-A)	494 (12.5%)	1 021 (25.8%)
	PrioriFI	**0 (0%)**	**0 (0%)**
CIFAR10-M	BinFI (A + I)	1 109 (0.2%)	4 024 (0.9%)
	StatFI (D-A)	2 699 (0.6%)	306 445 (66.5%)
	PrioriFI	**0 (0%)**	**0 (0%)**

5 Conclusion

PrioriFI is a more informed FI algorithm for edge NNs. It uses information gained during an FI campaign to dynamically determine which bit to flip next, from highest sensitivity to lowest. PrioriFI is generally effective at identifying sensitive bits faster in the nine edge NNs that we study in the paper. With PrioriFI, designers can quickly evaluate the robustness of different NN architectures under strict latency and resource constraints, codesigning fault-tolerant edge NNs faster and in a more informed way.

References

[1] [n. d.]. QKeras: a quantization deep learning library for Tensorflow Keras. https://github.com/google/qkeras. [Accessed 21-08-2025].

[2] 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.

[3] 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).

[4] 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).

[5] Giulio Borghello. 2020. Ionizing radiation effects on 28 nm CMOS technology. *CERN report "TID effects 28 nm"* (2020).

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

[7] 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).

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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).

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] Andy D Pimentel. 2016. Exploring exploration: A tutorial introduction to embedded systems design space exploration. *IEEE Design & Test* 34, 1 (2016), 77–90.

[25] 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.

[26] 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. doi:10.1023/A:1026543900054

[27] 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.

[28] Benjamin Carrion Schafer and Zi Wang. 2019. High-level synthesis design space exploration: Past, present, and future. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2019), 2628–2639.

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

[30] 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.

[31] 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.

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

[33] 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)*. 1–2. doi:10.23919/DATE56975.2023.10137033

[34] 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.

[35] 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. *ACM J. Auton. Transport. Syst.* 1, 3, Article 15 (July 2024), 27 pages. doi:10.1145/3665334

[36] 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.

[37] Jieun Yoo, Jennet Dickinson, Morris Swartz, Giuseppe Di Guglielmo, Alice Bean, Douglas Berry, Manuel Blanco Valentin, Karri DiPetrillo, Farah Fahim, Lindsey Gray, James Hirschauer, Shruti R Kulkarni, Ron Lipton, Petar Maksimovic, Corrinne Mills, Mark S Neubauer, Benjamin Parpillon, Gauri Pradhan, Chinar Syal, Nhan Tran, Dahai Wen, and Aaron Young. 2024. Smart pixel sensors: towards on-sensor filtering of pixel clusters with deep learning. *Machine Learning: Science and Technology* 5, 3 (aug 2024), 035047. doi:10.1088/2632-2153/ad6a00

A Experimental model information

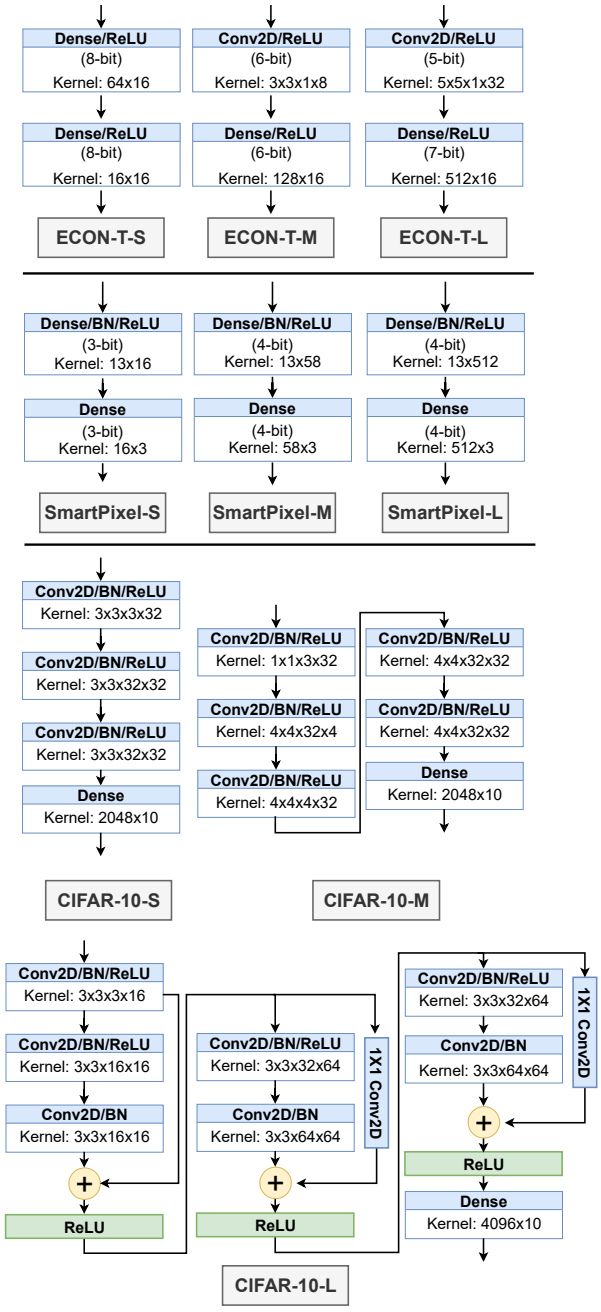


Figure 10. Edge NNs studied. Note that all CIFAR-10 models have been quantized to 8-bit fixed point.

Fig. 10 provides detailed layer-by-layer information on the layer type, quantization, and kernel size of each edge NN we study in the paper. All NNs were quantized via quantization-aware training using QKeras [1]. The ECON-T models are

based on the models presented in the FKeras paper [35]. The SmartPixel models are based on the models presented in the SmartPixel paper [37]. The CIFAR-10 models are based on the hls4ml-FINN submission to the MLPerf Tiny Benchmark [7].