On the Resilience of RTL NN Accelerators: Fault Characterization and Mitigation

Behzad Salami*†, Osman S. Unsal*, and Adrian Cristal Kestelman*†‡
*Barcelona Supercomputing Center (BSC), Barcelona, Spain.
†Universitat Politcnica de Catalunya (UPC), Barcelona, Spain.
‡IIIA - Artificial Intelligence Research Institute CSIC - Spanish National Research Council, Spain.
Emails: {behzad.salami, osman.unsal, and adrian.cristal}@bsc.es

***Abstract*—Machine Learning (ML) is making a strong resurgence in tune with the massive generation of unstructured data which in turn requires massive computational resources. Due to the inherently compute- and power-intensive structure of Neural Networks (NNs), hardware accelerators emerge as a promising solution. However, with technology node scaling below 10nm, hardware accelerators become more susceptible to faults, which in turn can impact the NN accuracy. In this paper, we study the resilience aspects of Register-Transfer Level (RTL) model of NN accelerators, in particular, fault characterization and mitigation. By following a High-Level Synthesis (HLS) approach, *first*, we characterize the vulnerability of various components of RTL NN. We observed that the severity of faults depends on both *i)* application-level specifications, i.e., NN data (inputs, weights, or intermediate) and NN layers and *ii)* architectural-level specifications, i.e., data representation model and the parallelism degree of the underlying accelerator. *Second*, motivated by characterization results, we present a low-overhead fault mitigation technique that can efficiently correct bit flips, by 47.3% better than state-of-the-art methods.**

I. Introduction

Machine learning models and in particular Neural Networks (NNs) are increasingly being used in the context of nonlinear ”cognitive” problems, such as natural language processing and computer vision. These models can learn from a dataset in the training phase and make predictions on a new, previously unseen data in the inference/prediction/classification phase with ever-increasing accuracy. However, the compute- and power-intensive nature of NNs prevents their effective deployment in resource-constrained environments, such as mobile scenarios. Hardware acceleration on Application Specific Integrated Circuits (ASICs) or Field Programmable Gate Arrays (FPGAs) offers a roadmap for enabling NNs in these scenarios [1], [2], [3], [4]. However, similar to general purpose devices, hardware accelerators are also susceptible to faults (permanent/hard and transient/soft), as a consequence of Single Event Upset (SEU), manufacturing defects, and below safe-voltage operations [5], [6]. The ever-increasing rate of these faults in nano-scale technology nodes, can directly impact the accuracy of NNs.

Traditionally, to perform early studies and apply further optimizations, application-specific hardware designs are modeled in different abstraction levels before the final in-silicon implementation. For instance, a hardware design can be modeled in, e.g., software-level behavioral simulator, functional level, Transaction-Level Model (TLM), Register-Transfer Level (RTL), and transistor-level. Among them, thanks to the evolution of High-Level Synthesis (HLS) tools to abstract low-level complexities of the hardware [7], the RTL model has received significant attention. This approach can lead to a decreased development time with early evaluation of the final design while conforming to final power, energy, performance, and resilience goals in comparison to the in-silicon ASIC/FPGA implementation. For instance, an HLS-based approach has been used in recent research to study the resilience of accelerators [8], [9], [10], [11]. In this paper, we also use an HLS-based approach to study accelerator resilience. However, we use the HLS approach to study the fault characterization and mitigation of the RTL model of NNs. Understanding this resilience behavior can provide an opportunity for the further resilience studies on the in-silicon NN accelerators. Main contributions of this work are summarized as follows:

- Fault Characterization: We perform an in-depth vulnerability study in the various components of the RTL NN accelerators against permanent and transient faults.
- Fault Mitigation: Motivated by the fault characterization experimental results, we present a low-overhead technique to mitigate faults by recovering corrupted bits, without any need for redundant data. The efficiency of the proposed method is by 47.3% better than the state-of-the-art methods.

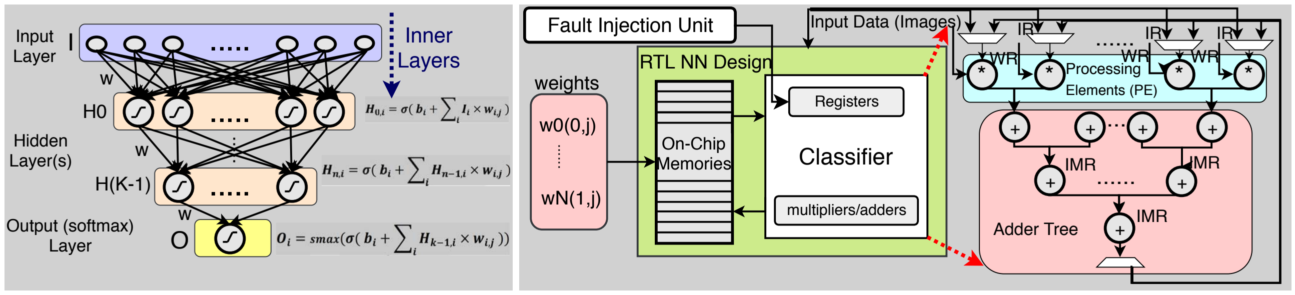
The rest of the paper is organized as follows. In Section II, the overall methodology, i.e., the RTL NN and fault model, is introduced. The fault characterization and mitigation studies are discussed in Section III, and Section IV, respectively. We review the previous work in Section V, and finally, the paper is summarized and concluded in Section VI.

II. Introducing the Overall Methodology

This section presents our methodology to conduct the resilience study, i.e., the architecture of the RTL NN accelerator and fault model.

A. The Architecture of RTL NN Accelerator

Specifications of the experimented RTL NN with a baseline configuration is summarized in TABLE I. Our study features a typical fully-connected NN that is also widely used in the structure of other NN models [2]. Our study targets the



(a) A Typicall Fully-Connected NN in the Inference Phase.

(b) RTL NN Accelerator Model.

Fig. 1: The overall methodology to resilience study of the RTL NN accelerator.

inference phase of NN since training is normally a one-time process; additionally, the inference is repeatedly performed to classify unknown data. As can be seen in Fig. 1(a), this NN model is composed of input, hidden, and output layers, where all adjacent layers are fully connected to each other. The first/last layer is the input/output layer and has one neuron for each component in the input/output vector. Between the input and output layers, there are single/multiple hidden layers. The interconnection between neurons of adjacent layers is determined based on a collection of weights and biases, whose values are tuned in the training phase. Each NN neuron uses an activation function to determine its output. Finally, in the output layer, a softmax function generates the final output of the NN. We perform our experiments on a 6-layer NN, i.e., ($\{L_i, i \in [0,5]\}$), one input, four hidden, and one output layer(s). The four hidden layer sizes are fixed at 1024, 512, 256, 128 while input and output layer sizes are benchmark-dependent (784, 54 and 2437 for input while 10, 8 and 52 for output layers for the three NN applications studied in this paper, i.e., MNIST [12], Forest [13], and Reuters [14], respectively.). Thus, there are five matrix multipliers among adjacent layers, i.e., ($\{Layer_j, j \in [0,4]\}$), where $Layer_j$ refers to the matrix multiplication of L_j and L_{j+1}. Among benchmarks, MNIST is a set of black and white digitized handwritten digits, each image composed of 784*8-bit pixels, the output infers the number from 0 to 9 (10 output classes), with 60000 training- and 10000 inference images. MNIST is the most widely-used by the ML community to evaluate the efficiency of novel NN methods. Hence, we use MNIST as the main benchmark to evaluate our resilience studies. To demonstrate the generality of experimental observations, we briefly present results for Forest and Reuters, as well.

We build the RTL NN leveraging Bluespec [15]. As can be seen in Fig. 1(b), our RTL design is composed of *i*) on-chip memories to accommodate weights, *ii*) different set of registers to latch data during the inference, *iii*) a set of multiplier-adder processing elements (PEs) and an adder-tree to perform the required matrix multiplications, and *iv*) a piece-wise linear model of the activation function. The explained model above is a typical model of NN accelerators, as surveyed in [16] for ASICs and in [17] for FPGAs. On this setup, input data are streamed through PEs in parallel to perform matrix multiplications in pipeline. Number of clock cycles to classify an input object item can be computed as a function of $|L_i|$ and the number of PEs, as shown in Eq. 1.[1]

TABLE I: Detailed specifications of the baseline RTL NN.

Neural Network (NN)	
Type	Fully-Connected Classifier
Topology (number of layers)	6L (1L input, 4L hidden, 1L output)
Per Layer Size (number of neurons)	(784, 1024, 512, 256, 128, 10)= 2714
Total Number of Weights	~1.5 million
Activation Function	Logarithmic Sigmoid (logsig)
Major Benchmark	
Name-Type	MNIST [12]- Handwritten Digits
Number of Images	Training: 60000, Inference: 10000
Number of Pixels per Image	28*28= 784
Number of Output Classes	10
Additional Benchmarks	
1. Forest	[13]
2. Reuters	[14]
Data Representation Model	
Type	16-bits Fixed-Point (FP)
Precision	Min sign and digit per layer (Fig. 2)
An Example Synthesize of RTL NN on FPGA	
FPGA Platform-Chip	VC707-Virtex7
Operating Frequency	100Mhz
BRAM Usage (Total: 2060)	70.8%
DSP Usage (Total: 2800)	8.6%
FF Usage (Total: 303,600)	3.8%
LUT Usage (Total: 607,200)	4.9%
Number of PEs	64

$$T = \frac{\sum_{i=0}^{N_l-2} |L_i| * |L_{i+1}|}{\#PEs} \tag{1}$$

It is important to note that our design is fully parameterizable on the number and size of NN layer and the number of PEs; also, it is fully synthesizable. For instance, the area utilization of synthesizing it on VC707, a Xilinx Virtex7 technology is shown in Table I. Furthermore, among pipeline stages of the RTL design, various registers are exploited to latch different types of NN data. At each cycle, a different data item is latched to these registers. As can be seen in Fig. 1(b), there are three types of registers, i.e., input registers (*IR*s),

[1]For instance, T of our baseline can be computed as: $T = \frac{784*1024+1024*512+512*256+256*128+128*10}{64} = 1490944$.

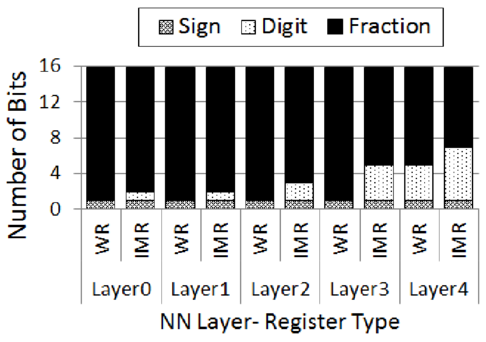


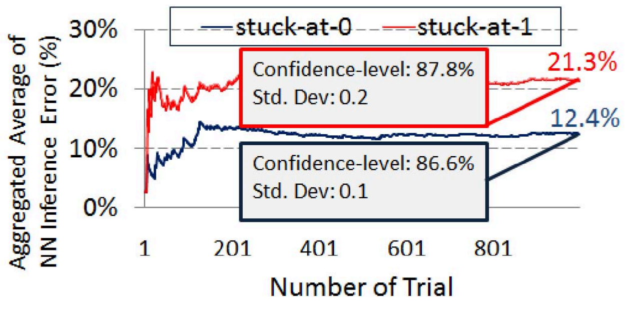
Fig. 2: Minimum precision to represent data of RTL NN, i.e., Weighs (WRs) and Intermediate (IMRs) Registers. (IRs are composed of only fraction components since input data are in [0,1) range.)

weight registers (*WR*s), and intermediate registers (*IMR*s), which are leveraged to latch input, weights, and intermediate data, respectively.

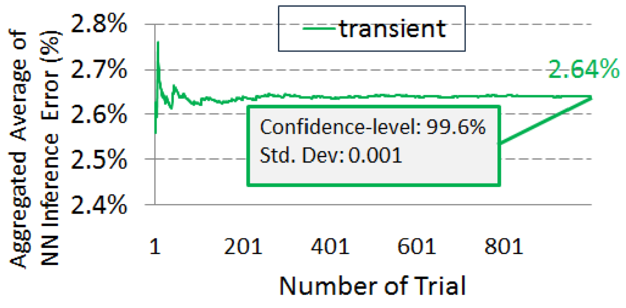
For experiments, we first export weights and biases of the trained NN that is performed off-line using a MATLAB implementation, initialize on-chip memories of the RTL design, and then start streaming 10000 input images to perform the inference. Also, for representing data, we use the fixed-point low-precision model. Note that lowering the precision of data is a common technique for applications in the approximate computing domain, in particular for NNs performing inference [18], to achieve power and performance efficiency with negligible accuracy loss. Following this approach, we use a per-layer minimum precision fixed-point model. The bit-width of data (input, weights, and intermediate) fixes to 16-bits, composed of the sign, digit, and fraction components. Toward this goal, with a pre-processing analysis, we extract the minimum bit-widths of the sign and digit components per layer, and the fraction component fills the rest of 16 bits. As we experimentally observed, this quantification does not lead to any considerable accuracy loss in comparison to a full-precision data model. Details of the number of bits per-layer are summarized in Fig. 2. Also, by following the default internal approach of Bluespec FixedPoint (FP) library, negative numbers (< 0) are represented in two's complement model.

B. Fault Model

As can be seen in Fig. 1, the fault injection unit is developed on top of the RTL NN classifier to manage fault injection into registers, i.e., IRs, WRs, and IMRs. It is important to note, other components of the RTL NN, e.g., on-chip memory and computing PEs are also susceptible to faults. However, many resilience solutions have been developed for these components, such as Error Correction Code (ECC) for memories [19] and time-staggered Razor shadow latches for PEs [20]; thus, we perform this study by fault injecting in registers, the main state-holding elements in the RTL design.



(a) Permanent Faults.



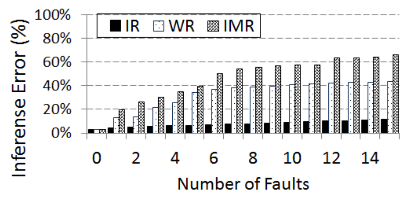
(b) Transient Faults.

Fig. 3: Repeating trials to achieve statistically significant results. (Shows results of single-bit fault trials)

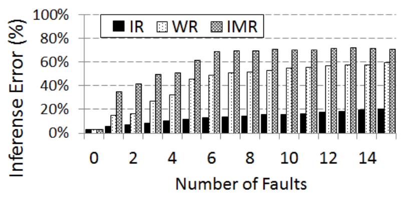
1) ***When***, ***Where***, *and* ***How frequently*** *are Faults Injected?:*

Our fault injection unit supports permanent (stuck-at-0 and stuck-at-1) and transient faults. Permanent faults can cause bits to permanently get stuck at 0 or 1, such as faults stemming from extremely low-voltage operation [21]. Thus, they result in bit flips, if and only if logical values of faulty bits are different than the values at which the physical bits are stuck. In contrast, transient faults such as radiation-induced faults [22], flip contents of bits for a few cycles (typically one cycle). Also, note that we evaluate the effect of both single-(single fault within a register) and multiple-bit (several faults within a register) faults to cover different fault rates. In short, we consider the following:

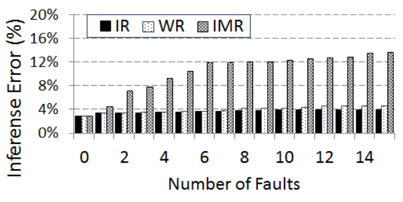
- ***When?*** Permanent faults are stuck at 0 or 1 for the whole inference cycles (T as defined in Eq. 1), from first to the last. In contrast, transient faults occur in a single random cycle within inference T cycles.
- ***Where?*** We inject faults into a randomly-selected set of bits of a register of the RTL NN.
- ***How frequently?*** To comprehensively study the impact of faults in the inference error, we repeat the fault injection for each input data item (for instance each image in MNIST). In other words, we generate a fault, and while input data is streaming into the classifier one-by-one, we inject the generated fault for each of the input data items, individually. This accelerated fault injection campaign allows us to quickly evaluate the impact of faults on all



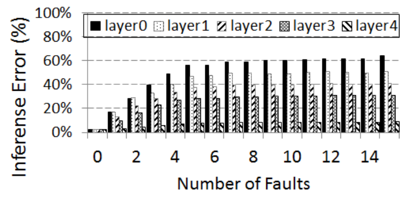
(a) NN Data: stuck-at-0



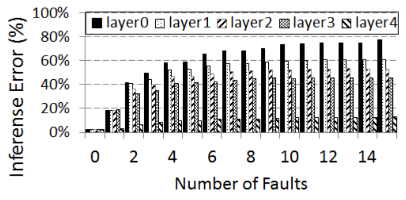
(b) NN Data: stuck-at-1



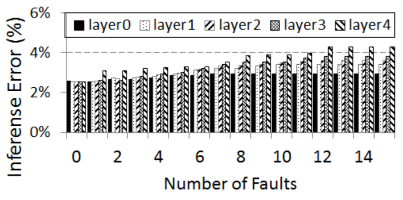
(c) NN Data: transient



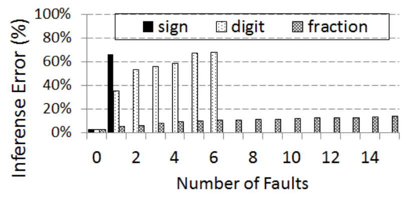
(d) NN Layers: stuck-at-0



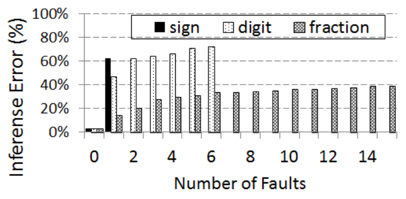
(e) NN Layers: stuck-at-1



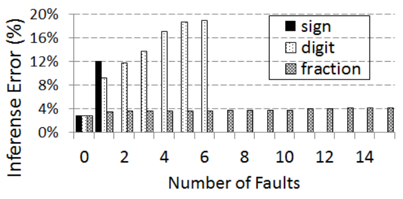
(f) NN Layers: transient



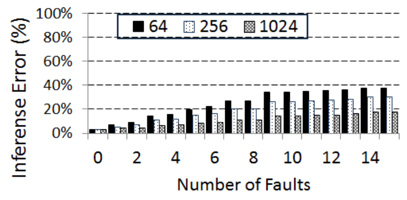
(g) FP Components: stuck-at-0



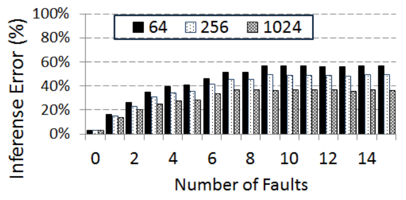
(h) FP Components: stuck-at-1



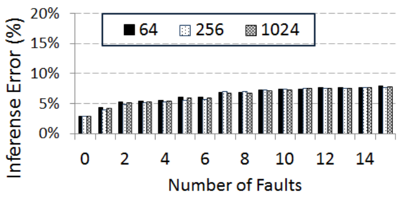
(i) FP Components: transient



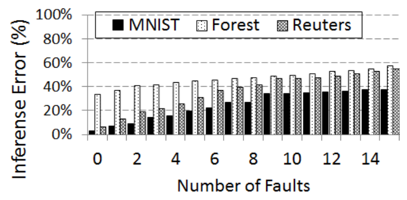
(j) Number of PEs: stuck-at-0



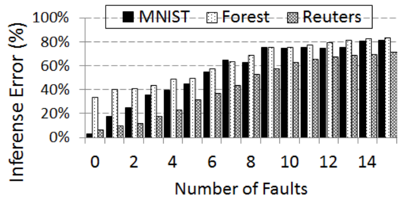
(k) Number of PEs: stuck-at-1



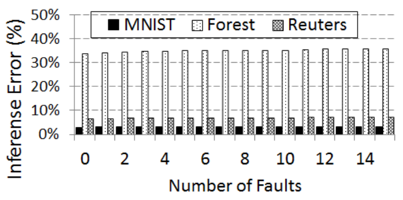
(l) Number of PEs: transient



(m) Datasets: stuck-at-0



(n) Datasets: stuck-at-1



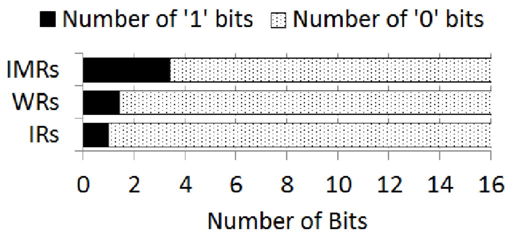
(o) Datasets: transient

Fig. 4: Fault characterization in RTL NN accelerator. (Different scales in the y-axis.)

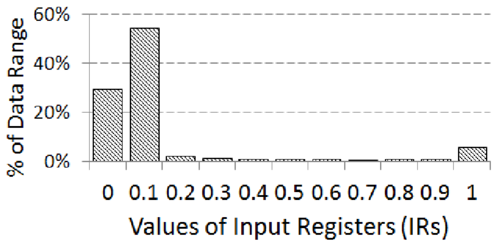
input data items.

To generate faults, we use the pseudo-random number generator library of Bluespec, i.e., Randomize, to select a random register, a random set of bits, and at a randomly-selected cycle (in the transient case). Later on, we inject the generated fault in the corresponding locations/cycles, while streaming the input data into the classifier, allow the completion of the inference for all input data, and finally, compute the inference error by comparing classified against the golden output data provided by the benchmark suite.

2) Statistical Fault Injection Methodology: To achieve statistically significant results, we repeat experiments multiple times. In each trial, a different fault (a random register, set of bits, and cycle in the transient case) is randomly generated and injected. Finally, after repeating the injection for multiple times, the final inference error is calculated as the median of all these trials. If trials are repeated for significantly enough times, this statistical fault injection approach [23] can lead to accurate results in comparison to the deterministic approach where all possible locations/cycles permutations are considered for the faults injection. Note that the deterministic approach is in



(a) Statistical sparsity analysis of Inputs (IRs), Weights (WRs), and Intermediate (IMRs) Registers.



(b) The statistical analysis in ranges of Input Registers (IRs) values.

Fig. 5: Histograms to show the sparsity of data of MNIST, used to justify; (a) why stuck-at-1 faults cause higher inference errors than stuck-at-0 faults, and (b) why *IMR*s are more vulnerable than *WR*s.

practice hard to follow since there is billions of possibilities of faults generation. To find enough number of trials note that it can be tuned according to the expected confidence level- the probability that the exact value is within a predefined error margin. Experimental results for an example single-bit fault injection case are shown in Fig. 3, in terms of the aggregated median error rate of RTL NN in the y-axis and the number of trials in the x-axis (up to 1000). As can be seen, with error margin 1%, 1000 trails lead to an acceptable confidence-level ($\sim$90%) with a negligible standard deviation. Hence, we perform all our experiments 1000 times and use the median of these trials to report in this paper.

III. Fault Characterization

In this section, we evaluate the sensitivity of various components of our RTL NN against faults; i.e., application-level components (NN data and NN layers) and architectural-level components (data representation model and the number of PEs). Toward this goal, we inject faults in these components, individually, by varying corresponding parameters of the baseline configuration as detailed in TABLE I. Experimental results of the fault characterization are shown in Fig. 4. In this figure, the x-axis represents the number of injected faults per input object (image in MNIST), which varies from 0 for the fault-free NN case, to 16 for the case where all bits of the NN register are faulty.

A. Overall Effect of Different Fault Types in RTL NN

We conduct the RTL NN fault characterization study in terms of the following fault categories:

- Permanent vs. Transient: As expected, permanent faults cause higher NN errors than transient faults. This observation is due to the persistence of permanent faults for the whole computation cycles (T as explained in Eq. 1), while transient faults manifest themselves for a single cycle within T cycles.
- Stuck-at-1 vs. Stuck-at-0: The permanent faults can be further categorized as being Stuck-at-1 or Stuck-at-0. As can be seen in Fig. 4, stuck-at-1 faults cause higher NN errors than stuck-at-0 faults. This observation is due to the sparsity of NN data, i.e., more '0' than '1' bits in the NN RTL registers, by on average 6.5X as can be seen in Fig. 5a.

B. Application-Level Fault Analysis

1) NN Data: The sensitivity of various NN data, i.e., inputs, weights, and intermediate data is evaluated by injecting faults in corresponding registers individually, i.e., IRs, WRs, and IMRs. Experimental results are shown in Fig. 4a, 4b, and 4c, for stuck-at-0, stuck-at-1, and transient cases, respectively. We observe that:

- IRs: Faults injection in IRs causes the relatively lowest inference error since they are composed of only fraction component.
- WRs and IMRs: In RTL NN, IMRs are the most vulnerable registers, due to two reasons; *first*, as can be seen in Fig. 2, IMRs have relatively the longest digit component, which is significantly sensitive against faults. *Second*, IMRs are used in the adder tree to maintain results of multipliers, as can be seen in Fig. 1. Therefore, any faults in IMRs is propagated to the next level of the adder tree without any coefficient. In contrast, WRs are inputs of multipliers and any fault in WRs is multiplied by IRs and then propagated. Nevertheless, however, IRs accommodate minimal values; as can be seen in Fig. 5a, 84.5% of them are in the [0, 0.1] range, which limits the fault propagation in WRs.

2) NN Layers: The sensitivity of NN layers, i.e., $\{Layer_j, j \in [0, 4]\}$ is evaluated by injecting faults in registers of these layers, individually. Experimental results are shown in Fig. 4d, 4e, and 4f, for stuck-at-0, stuck-at-1, and transient cases, respectively. We observe that:

- Permanent Faults: Permanent faults in inner layers of the RTL NN, i.e., closer to the output layer, cause relatively lower inference error. Due to the persistence of permanent faults for the whole per-layer cycles, this observation is the consequence of relatively less cycles in inner layers, proportional to layer sizes, as detailed in Table I.
- Transient Faults: In contrast, transient faults in inner layers have relatively more impact on the inference error, since, *first*, sizes of NN layers do not play any role for transient faults with momentary behavior, and *second*,

faults in inner layers have relatively less probability to be masked through the thresholding in the activation functions of earlier layers.

C. Architectural-Level Fault Analysis

1) Different Components of Data Representation Model: The sensitivity of various components of fixed-point data representation mode, i.e., sign, digit, and fraction is evaluated by injecting faults in corresponding components individually. Experimental results are shown in Fig. 4g, 4h, and 4i, for stuck-at-0, stuck-at-1, and transient cases, respectively. Note that for each component, the maximum number of injected faults is equal to the maximum bit-width of the corresponding components, as detailed in Fig. 2. We observe that sign, digit, and fraction components are in order, relatively more vulnerable, proportional to their positional significance, as expected.

2) Parallelism Degree (Number of PEs): The number of PEs can impact the fault propagation behavior, since with respect to Eq. 1, it inversely affects the number of inference cycles (T). T determines the data reuse degree, i.e., the number of registers reloads to accomplish the inference, which can be directly translated to the persistence of permanent faults. To empirically perform this experiment, we repeat the fault injection for the different number of PEs, i.e., 64, 256, and 1024. Fig. 4j, 4k, and 4l show experimental results in the presence of stuck-at-0, stuck-at-1, and transient faults, respectively. We observe that:

- Permanent Faults: In the presence of permanent faults, a relatively more PEs corresponds to proportionally reduced inference error. As said, it is the result of a relatively fewer number of clock cycles (T) and in turn, less persistence of permanent faults.
- Transient Faults: In contrast, in the presence of transient faults, the number of PEs does not play any role, due to the momentary behavior of transient faults.

D. Fault Characterization of Other NN Benchamrks

We repeat the fault characterization study in Forest and Reuters benchmarks, as well, by following the similar experimental methodology explained in Section II. We observed that a vast majority of observations on MNIST, as discussed in Sections III-A and III-B, is staying valid for Forest and Reuters, as well. Experimental results of the fault propagation in those datasets are shown in Fig. 4m, 4n, and 4o for stuck-at-0, stuck-at-1, and transient cases, respectively. Due to our experimental observations, the most important points are highlighted as follows:

- In our NN, the inherent fault-free inference error rate of MNIST, Forest, and Reuters are 2.56%, 38.7%, and 8.9%, respectively, very close to the state-of-the-art implementation [2].
- The rate of NN inference error increase of Reuter dataset in presence of additional permanent faults, is relatively more significant, since the size of the input layer of Reuter is considerably larger (2837 vs. 52/784), which in turn, leads to more persistence of permanent faults during the clock cycles (T).

List 1: Pseudo-code of proposed fault mitigation technique.

```
1: if (reg[N-1] & reg[N-2] are flipped) reg <= 0;
                              // WORD MASKING
2: else begin
3:    if (reg[N-1] is flipped) reg[N-1] <= reg[N-2];
                    // SIGN-BIT MASKING WITH MSB
4:    for(i in [N-2, 0]) if (reg[i] is flipped) reg[i] <= reg[N-1];
                              // BIT MASKING
5: end
```

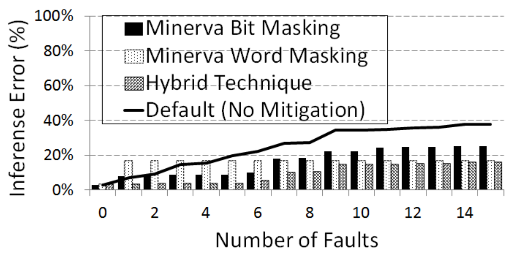
IV. Fault Mitigation

In this section, we introduce and evaluate an efficient and low-overhead fault mitigation technique in RTL NN, relying on the fault characterization results. The aim is to recover from faults without leveraging any redundant bits as is common for traditional fault correction mechanisms such as Triple Modular Redundancy (TMR) [26]. The proposed technique relies on the sparsity of NN data and accordingly, targets to mask faulty bits. This technique combines three individual mechanisms, i.e., *first*, Word Masking to set all bits of the corrupted register to '0', *second*, Bit Masking to mask faulty bit with the sign-bit within the faulty register, and *third*, Sign-bit Masking to mask the sign-bit with the Most Significant Bit (MSB). Among them, Word- and Bit Masking techniques are proposed the first time in Minerva [2]; however, they are used individually and with a poor efficiency to protect the sign-bit, which in fact is relatively the most vulnerable component. To alleviate this issue, we propose to mask the sign-bit with the MSB, since through a statistical analysis we observed that with the probability of 99,9% the sign-bit and MSB have the same logic value. This observation is the consequence of near-zero NN data and also two's complement data representation model, which finally leads to $sign\_bit = MSB$='0' for positive (> 0) and $sign\_bit = MSB =$'1' for negative near-zero data (< 0).

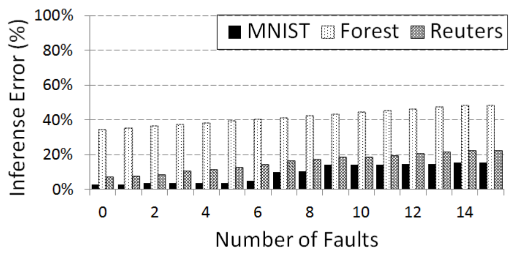
It is important to note that about the fault detection we follow same assumptions with Minerva, i.e., there is no limit on the number of faults that can be detected and also, information is available on which bits are affected. To achieve these assumptions, in Minerva, Razor shadow latches [40] are simulated that can detect faults by monitoring circuit delays. ASIC [2] and FPGA [39] implementation of Razor report 0.3% and 2.6% of area overheads, respectively; which shows the feasibility of exploiting this method in the synthesized version of our RTL NN, as well.

The pseudo-code of the proposed fault mitigation technique is shown in List. 1. In this pseudo-code, the bit-width of the register (*reg*) is assumed to be N, [N-1, 0], where **reg[N-1]** and reg[N-2] are referring to the **sign-bit** and MSB, respectively. As can be seen, the proposed technique is composed of three sub-methods:

- Sign-Bit Masking with MSB: We use MSB as the mask of the sign-bit since through an experimental analysis we



(a) Evaluating different fault mitigation techniques (on MNIST).



(b) Enhanced hybrid technique on different datasets.

Fig. 6: Evaluating fault mitigation techniques (shown for stuck-at-0 case as similar efficiency observed for other types).

observed these bits have same logic, with the probability of 99.9%.

- Bit Masking: Then, we apply the Bit Masking technique to recover non-sign-bit flips, by using sign-bit as the mask.
- Word Masking: And finally, if MSB is itself also flipped, we reset the register to '0'.

We evaluate the proposed technique in our RTL NN and compare it with the individual Word- and Bit Masking techniques. Experimental results of the evaluation of these fault mitigation techniques are shown in Fig. 6a. As can be seen:

- Minerva Bit Masking: The efficiency of this technique is relatively the worst, since not only it does not have any mechanism to protect the sign-bit but also masks other faulty bits with sign-bit, which can be faulty.
- Minerva Word Masking: This technique leads to a constant inference error, independent of the number of detected faults since the faulty register is reset to '0', when at least one fault is detected.
- Proposed Enhanced Hybrid Technique: This technique shows relatively the best performance to correct faulty bits and achieve the lowest inference error since it takes the advantage of both Bit- and Word Masking techniques and also, complement it by using MSB to mask the sign-bit faults.

In short, our Enhanced Hybrid Technique mitigates faults and achieve by 47.3% and 44.1% lower NN inference error than Minerva Bit- and Word Masking techniques, respectively.

We apply the proposed enhance hybrid fault mitigation technique on Forest and Reuters datasets, as well. The efficiency of the proposed fault mitigation method is shown in Fig. 6b. By comparing these results against the default experiments without any protection in Fig. 4m, it can be observed that the proposed method can cover faults in all tested datasets since the sparsity of data is an inherent feature of these benchmarks and it is also the base of the proposed technique.

V. Related Work

In this section, we review recent works on the resilience of NNs and highlight our contributions.

A. *Different Methodologies to Study NN Resilience*

It has been shown that NNs are inherently resilient [27], [28]; however, the ever-increasing fault rate in nano-scale technology nodes necessitates further studies in this area to explorer better trade-off of reliability, power, energy, and performance. Hence, in recent years NN resilience is studied with different approaches, e.g., software-level simulations or theoretical analyzes [29], [30], SPICE simulations [2], [31], [32], [33], [34] and experimenting on the real hardware operating on low-voltage regimes [35], [36], [37], [38]. Among them, it is evident that software-level simulations and theoretical analyzes lack the information of the underlying hardware platform and are relatively less precise. In contrast, SPICE-based studies are more precise; however, these studies require significant circuit-level efforts. We aim to have the best of both worlds: to have the flexibility and simplicity of software-level fault-injection with the precision of circuit-level implementations. Thanks to the evolution of HLS tools, the precise modeling of the underlying hardware has been facilitated, which provides an opportunity to perform such studies comprehensively with an accuracy close to the real hardware. This approach is also followed by some recent works to study the resilience of several other applications [8], [9], [10], [11].

B. *Fault Characterization on NNs*

The impact of faults in different models of NNs have been studied in a wide body of recent work as surveyed in [27]. Among the most relevant existing works, Minerva [2] performs a characterization on the sparsity of data and analyzes the efficiency of leveraging fixed-point data representation model. In the same line, [28] studied the vulnerability of various layers of NN. Also, recently [31] studied the fault propagation in an ASIC model of NN focused on the vulnerability of different NN layers. However, these works do not present comprehensive characterization studies, whereas our paper comprehensively studies the sensitivity analysis of both application- (NN data and NN layers) and architectural-level (parallelism degree and data representation model) components. This comprehensiveness is the consequence of using HLS tools to facilitate the RTL modeling of the NNs.

C. *Fault Mitigation on NNs*

To mitigate faults several general techniques are proposed in different domains such as TMR [26], Razor [40], [20], ECC using Hamming code [19], Hardware Transnational Memory

(HTM) [41], among others. These techniques can be potentially customized to mitigate faults of NNs, as well; however, with timing, area, or power costs. Also, techniques adapted for NNs are surveyed in [42], such as explicit redundancy, retraining, and modifying learning/inference phases. In this paper, instead of costly fault tolerance operations, we present application-aware fault mitigation in NNs, which does not require to exploit any redundant data bits or considerable additional overheads. In the same line, Minerva [2] has also proposed two fault mitigation techniques, i.e., bit masking and word masking, which rely on the sparsity of NN data. In our paper, we use these techniques as baseline comparison cases. We explored their efficiency issues when the sign-bit is corrupted and accordingly, presented a more efficient technique to be effective in such cases, as well.

VI. Conclusion and Future Work

We empirically investigated an RTL hardware-aware design of NN accelerator from the resilience perspective, i.e., fault characterization and mitigation. We comprehensively characterized the vulnerability of various components of the design, in the presence of permanent and transient faults. Our experiments experimentally shows the severity of RTL NN components against different types of faults. Relying on the characterization results, we proposed a low-overhead fault mitigation technique to correct corrupted bits of RTL NN efficiently. Our proposed method can mitigate faults, by 47.3% better than state-of-the-art methods. We plan to perform fault characterization and mitigation study on more advanced RTL NN models, e.g., Convolutional NNs (CNNs) and Recursive NNs (RNNs), as well.

Acknowledgment

We thank Pradip Bose, Alper Buyuktosunoglu, and Augusto Vega from IBM Watson for their contribution to this work. The research leading to these results has received funding from the European Union's Horizon 2020 Programme under the LEGaTO Project (www.legato-project.eu), grant agreement n° 780681.

References

[1] Li, H., et al. "A high performance FPGA-based accelerator for large-scale convolutional neural networks", in *FPL*, 2016.

[2] Reagen, B., et al. "Minerva: Enabling low-power, highly-accurate deep neural network accelerators." in *ACM SIGARCH Computer Architecture News*, 2016.

[3] Chen, Y. H., et al. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks", in *IEEE Journal of Solid-State Circuits*, 2017.

[4] Andri, R., et al. "Yodann: An architecture for ultralow power binary-weight cnn acceleration", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[5] B. Salami, et al. "Fault Characterization Through FPGA Undervolting", in *FPL*, 2018.

[6] B. Salami, et al. "A Demo of FPGA Aggressive Voltage Downscaling: Power and Reliability Tradeoffs", in *FPL*, 2018.

[7] Nane, R., et al. "A survey and evaluation of fpga high-level synthesis tools", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[8] Tambara, L. A., et al. "Analyzing reliability and performance trade-offs of HLS-based designs in SRAM-based FPGAs under soft errors", in *IEEE Transactions on Nuclear Science*, 2017.

[9] Lee, G., et al. "TLegUp: A TMR Code Generation Tool for SRAM-Based FPGA Applications Using HLS". in *FCCM*, 2017.

[10] Tonfat, J., et al. "Soft error susceptibility analysis methodology of HLS designs in SRAM-based FPGAs". in *MICPRO*, 2017.

[11] dos Santos, A. F., et al. "Applying TMR in hardware accelerators generated by high-level synthesis design flow for mitigating multiple bit upsets in SRAM-based FPGAs", in *ARC*, 2017.

[12] Y. LeCun, et al. "Gradient-based learning applied to document recognition", in *IEEE Proceeding*, 1998.

[13] J. A. Blackard. "Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types", PhD thesis, *Colorado State University*, 1998.

[14] A. Cardoso-Cachopo. "Improving Methods for Single-label Text Categorization", PhD thesis, *Universidade Tecnica de Lisboa*, 2007.

[15] Bluespec Co: http://bluespec.com/

[16] Sze, V., et al. "Efficient processing of deep neural networks: A tutorial and survey", in *Proceedings of the IEEE*, 2017.

[17] K. Guo, et al. "A Survey of FPGA Based Neural Network Accelerator", *arXiv:1712.08934*, 2017.

[18] S. Gupta, et al. "Deep learning with limited numerical precision", in *ICML*, 2015.

[19] G. Miller, et al. "Single-event upset mitigation for xilinx FPGA block memories", *XILINX Application Note*, 2007.

[20] E. A. Stott, et al. "Timing fault detection in FPGA-based circuits", in *FCCM*, 2014.

[21] K. K. Chang, et al. "Understanding reduced-voltage operation in modern DRAM devices: Experimental characterization, analysis, and mechanisms", in *Measurement and Analysis of Computing Systems*, 2017.

[22] Wirthlin, M., et al. "The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets", in *FCCM*, 2003.

[23] Leveugle, R., Calvez, et al. "Statistical fault injection: Quantified error and confidence", in *DATE*, 2009.

[24] MathWorks, https://es.mathworks.com/help/nnet/ref/satlin.html

[25] MathWorks, https://es.mathworks.com/help/nnet/ref/logsig.html

[26] M. J. Wirthlin. "Improving the reliability of FPGA circuits using triple-modular redundancy (TMR) & efficient voter placement". in *FPGA*, 2004.

[27] C. Torres-Huitzil, et al. "Fault and Error Tolerance in Neural Networks: A Review", in *IEEE Access*, 2017.

[28] O. Temam. "A defect-tolerant accelerator for emerging high-performance applications", in *ISCA*, 2012.

[29] E. B. Tchernev, et al. "Investigating the fault tolerance of neural networks", in *Neural Computation*, 2005.

[30] Phatak, D. S., et al. "Complete and partial fault tolerance of feedforward neural nets", in *IEEE Transactions on Neural Networks*, 1995.

[31] G. Li, et al. "Understanding error propagation in deep learning neural network (DNN) accelerators and applications", in *SC*, 2017.

[32] J. Zhang, et al. "ThUnderVolt: Enabling Aggressive Voltage Underscaling and Timing Error Resilience for Energy Efficient Deep Neural Network Accelerators", in *DAC*, 2018.

[33] J. Zhang, et al. "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator." in *VTS*, 2018.

[34] K. Sung, et al. "MATIC: Learning around errors for efficient low-voltage neural network accelerators." in *DATE*, 2018.

[35] B. Salami, et al. "Comprehensive Evaluation of Supply Voltage Underscaling in FPGA on-chip Memories", in *Micro*, 2018.

[36] P. N. Whatmough, et al. "14.3 a 28nm SOC with a 1.2 ghz 568nj/prediction sparse Deep-Neural-Network engine with¿ 0.1 timing error rate tolerance for IOT applications", in *ISSCC*, 2017.

[37] L. Yang, et al. "SRAM voltage scaling for energy-efficient convolutional neural networks", in *ISQED*, 2017.

[38] L. Yang, et al. "Approximate SRAM for Energy-Efficient, Privacy-Preserving Convolutional Neural Networks", in *ISVLSI*, 2017.

[39] Stott, E., et al. "Timing fault detection in FPGA-based circuits", in *FCCM*, 2014.

[40] D. Ernst, et al. "Razor: A low-power pipeline based on circuit-level timing speculation", in *MICRO*, 2003.

[41] G. Yalcin, et al. "FaulTM: error detection and recovery using hardware transactional memory", in *DATE*, 2013.

[42] Torres-Huitzil, C., et al. "Fault tolerance in neural networks: Neural design and hardware implementation", in *ReConFig*, 2017.