Journal of Systems Architecture 104 (2020) 101689

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A survey on modeling and improving reliability of DNN algorithms and accelerators☆

Sparsh Mittal

Department of Computer Science and Engineering, IIT Hyderabad, India

ARTICLE INFO

Keywords:
Review
Deep neural networks
Permanent fault
Transient fault
Deep learning
Fault-injection

ABSTRACT

As DNNs become increasingly common in mission-critical applications, ensuring their reliable operation has become crucial. Conventional resilience techniques fail to account for the unique characteristics of DNN algorithms/accelerators, and hence, they are infeasible or ineffective. In this paper, we present a survey of techniques for studying and optimizing the reliability of DNN accelerators and architectures. The reliability issues we cover include soft/hard errors arising due to process variation, voltage scaling, timing errors, DRAM errors due to refresh rate scaling and thermal effects, etc. We organize the research projects on several categories to bring out their key attributes. This paper underscores the importance of designing for reliability as the first principle, and not merely retrofit for it.

1. Introduction

In recent years, DNN[1] algorithms have surpassed human-level accuracy for many cognitive applications. This trend has motivated the researchers to use DNN algorithms for many mission-critical applications, such as health, defense, space, finance, drones, and autonomous driving. In the design of DNN algorithms, the primary focus of optimization has been accuracy and throughput. Recent improvements in the design of DNN accelerators have addressed the concerns of energy efficiency also. However, reliability, which is a crucial metric of optimization, especially in the mission-critical applications, has not been given as much importance.

No doubt, DNN accelerators enjoy a certain degree of fault-tolerance due to the fault-tolerance of DNN algorithms themselves [1]. For example, the ReLU, normalization and max-pooling layers help in masking the impact of faults [2–6]. Also, retraining process adapts the weights to minimize the impact of faults [7–13]. Skipping some faulty connections between neurons does not significantly impact reliability since DNNs already use techniques such as dropout and dropconnect [14]. Narrow-width and quantized data-values such as 8b integers or binary values generally have higher immunity than wide FP values such as FP64/FP32 since their smaller dynamic range clamps the magnitude of a fault [4,15]. Hence, DNNs using such low-precision/quantized values show high robustness to faults. Further, since the output of DNN is generally a ranking, an error which changes the confidence scores but does not change the ranking of predicted class labels is not critical [4].

However, the simple intuition discussed above is not sufficient for fully reasoning about the reliability of state-of-the-art DNNs which have higher depth, width, and more complex-design than their predecessors. The vulnerability values of different DNNs and even different layers of a single DNN vary widely [16]. The vulnerability also depends on several other factors such as the precision, kernel size, data-structure (e.g., weight vs. activation) and even the value of the activation such as whether it is positive or negative, etc. [4,17–19]. Further, several techniques that improve other metrics such as energy/memory-footprint harm reliability. For example, to achieve high data-reuse, the memory in DNN accelerators is accessed repeatedly, and due to this, a faulty value may be reused several times.

☆ Support for this work was provided by Science and Engineering Research Board (SERB), India, award number ECR/2017/000622 and “Semiconductor research corporation”.

E-mail address: sparsh@iith.ac.in

[1] The following acronyms are used frequently in this paper: architectural/program vulnerability factor (AVF/PVF), artificial/convolution/deep/recurrent neural network (ANN/CNN/DNN/RNN), backward/forward propagation (BWP/FWP), batch normalization (BN), bit error rate (BER), convolution (CONV), complementary/n-type/p-type metal-oxide semiconductor (CMOS/NMOS/PMOS), detected unrecoverable error (DUE), error correcting code (ECC), failure in time (FIT), floating/fixed point (FP/FxP), fully-connected (FC), generalized matrix multiplication (GEMM), least/most significant bit (LSB/MSB), multi-layer perceptron (MLP), multiply-and-accumulate (MAC), near-threshold voltage (NTV), processing engine (PE), process variation (PV), register file (RF), silent data corruption (SDC), “single error correction double error detection” (SECDED), soft error (SE), tensor processing unit (TPU), timing error (TE), triple modular redundancy (TMR).

https://doi.org/10.1016/j.sysarc.2019.101689
Received 26 August 2019; Received in revised form 14 October 2019; Accepted 23 November 2019
Available online 28 November 2019

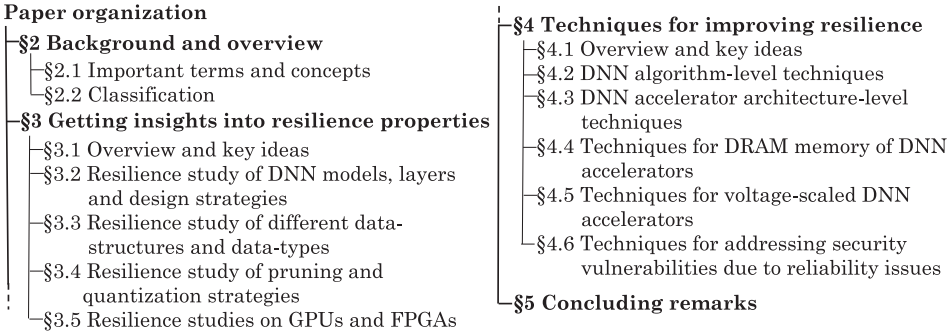


Fig. 1. Organization of the paper .

Conventional techniques for improving reliability, such as redundancy or re-computation do not take into account the characteristics of DNN accelerators and algorithms. Hence, they may provide under-protection or over-protection, and also incur very high overheads. For example, schemes based on detecting bit-wise differences detect faults that are masked by the DNN anyway [4]. The number of static instructions in a DNN accelerator is very few, e.g., TPU has nearly 12 instructions. Hence, replicating even one static instruction leads to a high penalty. As another example, the memory bandwidth demands of DNN accelerators are already high (e.g., 250 GB/s for DianNao) and adding redundant storage makes it even higher.

Also, with process technology scaling, the frequency of multi-bit errors is rising [20]; however, most conventional reliability solutions work under the assumption of single-bit errors. Even TMR can handle the fault in only one of its modules since its majority voting approach fails when more than one modules are faulty. The techniques for correcting multi-bit errors such as multi-bit ECC have high overhead and become infeasible in latency-critical applications [21]. Finally, under aggressive optimizations such as voltage scaling and DRAM refresh rate scaling, the conventional reliability techniques may no longer be sufficient. These challenges and factors call for a thorough and dedicated study and optimization of the reliability of DNN algorithms and accelerators. Several recent works seek to fulfill this crucial need.

Contributions: In this paper, we present a survey of techniques for characterizing and improving the resilience of DNN algorithms and accelerator architectures. The reliability issues we cover include soft/hard errors arising due to process variation, voltage scaling, timing errors, DRAM errors due to refresh rate scaling and thermal issues, etc. Fig. 1 provides an overview of the paper. Section 2 provides the relevant background, and a classification of the research works on crucial parameters. Section 3 reviews the works that study the impact of various design heuristics on DNN reliability and that provide insights into the reliability characteristics of DNN layers, data-structures, and datatypes. Section 4 reviews techniques for improving DNN reliability. In these sections, we mainly focus on qualitative ideas to gain insights and include only selected numerical results. Section 5 concludes the paper with a discussion of future work.

Scope: To make a concise presentation, we limit the scope of this paper as follows. We include works that focus on reliability issues and their implications on performance, energy efficiency, and security. We do not include works that focus only on performance/energy/security optimizations to DNNs. We review reliability issues in neural network algorithms/architectures and not in other machine learning algorithms. We do not include works for improving the reliability of DNN engines that use emerging memories such as memristor [22]. This paper will be useful for researchers and application-designers in the field of deep learning, computer architecture, and circuit-designers. This paper is also much more comprehensive and detailed than the previous summaries [23,24].

2. Background and overview

2.1. Important terms and concepts

We now discuss relevant terms and concepts which will be useful throughout this paper.

Floating-point and fixed-point formats: Fig. 2(a)–(b) show the IEEE-754 format for single-precision (FP32) and half-precision (FP16). Following [4], a fixed-point (FxP) format is shown as *K*b_rb*F*, which, from left to right, has 1 sign bit, $K - F - 1$ integer bits and F fraction bits. This format is shown in Fig. 2(c).

Since some techniques [2,18,25] work based on the value of nine MSBs (i.e., 31st-23rd bits) of FP32 representation, Table 1 shows their value for power-of-two numbers in the range of $[2^{11}, 2^{-15}]$.

Fault taxonomy: FIT is defined as a failure rate of 1 per billion hours. Fig. 3 shows the propagation of a fault in a bit to the program-level error [26]. It is noteworthy that on using an error-detection scheme such as ECC, some benign faults may turn into false DUEs, and SDCs may turn into true DUEs [27]. For a detailed discussion on fault taxonomy, we refer the reader to prior work [28].

AVF and PVF: AVF shows the fraction of bit-level faults that propagate to program-level result. PVF shows the fraction of faults injected at the instruction level that impact the program result.

Fault-injection approaches: To quantitatively model the reliability of a computing system or a processor component, their behavior under faults is studied. For fault-injection, two broad approaches have been used, which are complementary to each other. (1) In software fault-injection, faults are emulated in the software based on a certain fault model. For example, a transient fault due to particle strike may be modeled as the flipping of a fraction of bits. Similarly, a permanent fault such as a stuck-at-0 memory cell may be modeled by always returning the value '0' on reading that cell. Software fault-injection allows performing repeatable and controlled experiments and also studying future architectures, which may not be currently available. However, a limitation of fault-injection studies is that deriving statistically-sound results requires performing many experiments.

(2) In beam-radiation experiments, energetic particles such as neutrons are irradiated on the computing system. Since the intensity of this neutron flux is several orders of magnitude higher than that of terrestrial flux, these experiments emulate thousands of years of normal operation. Also, the results from these experiments are independent of the fault model assumed. However, these experiments are costly, and beam-radiation facilities are available only in few labs. Further, the results of beam studies depend on the amount of exposed component and fault propagation, whereas the results of fault-injection depend only on fault propagation.

Near-threshold voltage computing (NTV): NTV refers to operating the transistor very close to but above its threshold voltage [29]. It saves energy at the cost of performance degradation, increased impact of process variation [30] and fault-rate.

Table 1
2^k for $k \in [11, -15]$ and 31st-23rd bits of corresponding FP32 representation.

2^k	31st-23rd bits	2^k	31st-23rd bits	2^k	31st-23rd bits	2^k	31st-23rd bits	2^k	31st-23rd bits
2^{11}	0 10001010	2^5	0 10000100	2^{-1}	0 01111110	2^{-7}	0 01111000	2^{-13}	0 01110010
2^{10}	0 10001001	2^4	0 10000011	2^{-2}	0 01111101	2^{-8}	0 01110111	2^{-14}	0 01110001
2^9	0 10001000	2^3	0 10000010	2^{-3}	0 01111100	2^{-9}	0 01110110	2^{-15}	0 01110000
2^8	0 10000111	2^2	0 10000001	2^{-4}	0 01111011	2^{-10}	0 01110101		
2^7	0 10000110	2^1	0 10000000	2^{-5}	0 01111010	2^{-11}	0 01110100		
2^6	0 10000101	2^0	0 01111111	2^{-6}	0 01111001	2^{-12}	0 01110011		

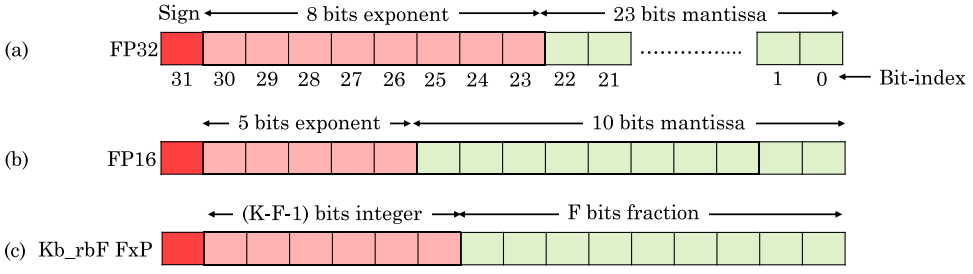


Fig. 2. (a) IEEE-754 FP32 format (note the bit-index convention used for its 32 bits) (b) IEEE-754 FP16 format (c) A generic *K*b_rb*F* FxP format.

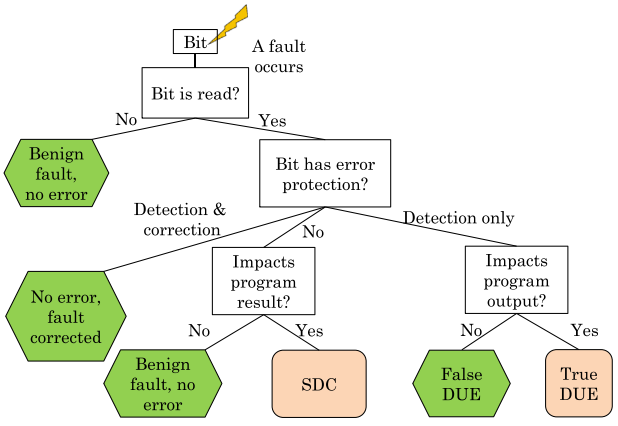


Fig. 3. Different cases of propagation of a fault in a bit to application-level error [26].

Timing error (TE): TEs arise from factors such as aging, crosstalk, process variation, timing delays, etc. [31]. Further, the use of frequency and voltage scaling can also lead to TE. For example, when the clock frequency is high, but the voltage is low, TEs arise due to lack of adequate timing margin. With ongoing process technology scaling, the rate of TEs is increasing.

Razor technique: The Razor technique [32] uses a double sampling flip-flop for detecting timing errors in a pipeline stage. The instruction which sees the error is replayed at a lower clock frequency for preventing further timing error.

Unstructured and structured pruning of DNNs: The unstructured pruning scheme prunes arbitrary weights. This may bring large reduction in the model size, however, the weight matrix and memory accesses become irregular. Hence, on high-performance parallel processors such as GPUs and multicore CPUs, the performance benefits of unstructured pruning may not be commensurate with the reduction in model size. In comparison, structured pruning prunes the weights in a hardware-aware manner. This retains the regularity of memory access pattern which allows achieving high performance on parallel processors.

LRN layer: LRN layer implements the "lateral inhibition", which refers to ability of an excited neuron to reduce the activity of its neighbors. This enhances the local contrast which allows using the locally maximum pixels value as the excitation for subsequent layers. It also

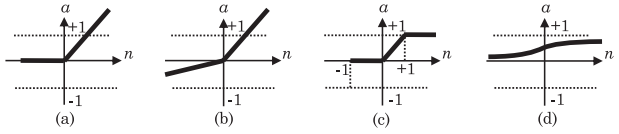


Fig. 4. Activation functions (a) ReLU (b) parametric ReLU (PReLU) (c) "positive saturating linear" (satlin) [33] (d) "logarithmic sigmoid" (logsig) [34] .

dampens the responses that are uniformly large in a local neighborhood and thus, normalizing them reduces the values of all the neurons.

Activation functions: Fig. 4 shows the activation functions used by different works summarized in this survey.

2.2. Classification

Table 2 shows the type of DNN models used for evaluation such as CNN, MLP and RNN. Further, it shows the specific DNNs and datasets used for evaluation. It is clear that most techniques have studied CNNs designed for smaller datasets such as MNIST and CIFAR-10. Much less number of works have studied DNNs such as AlexNet, VGG, ResNet etc. which are designed for large datasets such as ImageNet. Future works should focus on studying the reliability of state-of-the-art complex DNNs trained on large-scale datasets. Table 2 also lists the processing unit and processor component whose reliability has been evaluated. Finally, Table 2 shows the evaluation metrics used by different works. Evidently, reliability issues have important energy and security ramifications.

Table 3 shows the types and sources of faults studied by different works. It also shows the fault-injection approach used by them.

3. Getting insights into resilience properties

Here, we first summarize the key ideas of different works (Section 3.1). Then, we review works that study reliability properties of DNN models, layers and design strategies (Section 3.2), different data-structures and data-types (Section 3.3) and pruning and quantization strategies (Section 3.4). We then summarize the studies performed on GPUs and FPGAs (Section 3.5).

3.1. Overview and key ideas

Table 4 shows the DNN components whose reliability has been studied and the factors whose impact on reliability has been investigated.

Table 2
A classification based on evaluation context, platform and metrics.

NN used for evaluation	
CNN	nearly all
RNN	[16,35]
MLP	[5,7,8,11,12,14,17,19,36,37]
Specific DNN model used for evaluation	
Image classification	AlexNet [2,4,9,12,14,18,19,25,35,37–39], VGG [2,6,9,15,16,25,35,37,39–42], ResNet [2,6,16,18,43,44], LeNet [2,9,11,15,16,36,42,45], SqueezeNet [18,46] GoogleNet [18,41], InceptionV3 [2,6], NiN [4], DenseNet [2], MobileNet [44]
Object detection	YOLO [27,43,45,47], Faster RCNN [43]
Image understanding	LRCN [35]
Datasets used for evaluation	
ImageNet	[2,4,6,9,14,16,18,25,35,38–42,46]
MNIST	[2,4,5,10–12,14–17,19,37,45–52]
CIFAR-10	[2,5,11,15–17,19,25,40,44,46,48]
Reuters	[14,48,49,51]
Others	PASCAL VOC2007 [12], TIMIT [12,14], Forest [49,51], WebKB [49], 20NG [49], SVHN [5], German traffic sign recognition [37]
Processing unit used for evaluation (other works are agnostic to processing unit)	
GPU	[10,27,47]
FPGA	[9,38,47,52,53,53,54]
TPU	[12,14,42,48]
Processor component whose reliability is studied	
DRAM (main memory)	2D DRAM [2,5,18], die-stacked DRAM [25,35]
SRAM (on-chip memory)	[50]
Register	[51]
Adder and/or multiplier	[8,36,55]
Evaluation metric	
Accuracy	nearly all
Energy	[5,8,14,18,42,44,49,56]
Security	[2,5]

Table 3
A classification based on source and types of faults.

Single/multi-bit fault	
Multi-bit faults	[46,55,57]
Single-bit fault	nearly all
Type of fault	
Transient (soft)	[2,4–6,9,14,15,17–19,25,35–37,39,41,42,46,46,48–52,55,57,58]
Permanent (hard)	[8,10,12,16,45,51,58]
Reason for fault	
Process variation	[7,11]
Timing error	[7,13,14,36,56]
Voltage scaling (and near-threshold computing)	[14,16,48,50,59,60]
Analog/mixed-signal operation	[11,61,62]
Faults in DRAM	refresh rate scaling [18,41], thermal issues [25,35], row hammer [2], restore errors due to scaling of tRAS [5], charged particle strike [25]
Fault-injection approach	
Beam radiation	[3,27,47,52–54]
Software fault injection	nearly all others

Table 4
Factors/strategies whose impact on reliability is studied and components whose reliability is studied.

FC layers	[3,4,11,15,16,37,51,52]
Pooling layers	[3,4]
Dropout/dropconnet schemes	[2,14]
Deep vs. shallow (or small vs. large) DNNs	[2,11,14,17,63]
Per-layer analysis	[3,4,10,14,16,19,42,44,45,49,51–53,57]
Different data-structures (e.g., weight, activations/input)	[16,18,19,49,51,57]
Datatypes with different precisions (FP16, FP32, FP64 and FxP)	[3,4,15,16,47]
Different bit positions (e.g., exponent and mantissa)	[2–4,12,16,18,25,38,41,57]
Memory vs. compute/control elements	[4,27,57]
Direction of bit-flips ($0 \rightarrow 1$ vs $1 \rightarrow 0$)	[2,4,10,25,39,51]
Activation layer and activation functions	[2,4,8,49,51]
Positive and negative numbers	[2]
Different kernel size (e.g. 3x3 vs. 7x7)	[17]
Number of PEs in DNN accelerator	[51]

We now summarize the key observations of different works.

(1) *Impact of increasing BER:* Several DNNs show a negligible loss in accuracy as long as the BER is below a certain threshold, but beyond this threshold, they show an exponential reduction in accuracy [8,14,36,55,57]. With increasing BER, not only more data-values of a DNN become faulty, but the magnitude of change in data-values also becomes high. Larger the magnitude of change in value due to a fault, higher is the probability of output error [4,8,12].

(2) *Metrics used by different works:* For quantitative evaluation, researchers have used different metrics, which are complementary since they provide different insights and are applied in different scenarios/DNNs. Some of the metrics are (1) classification accuracy (for image classification DNNs) (2) ability to detect the object (for object detection DNNs) [3]. (3) The highest BER at which accuracy loss remains zero [15,16] (4) percentage of parameters that are vulnerable (i.e., a bit-flip in it degrades accuracy severely) [2] (5) percentage of trials that reach a certain accuracy [17] (this is used when faults are introduced at random locations or in random weights and a large number of trials are performed to find statistically meaningful results) (6) the number of simultaneous bit flips required for making a DNN unstable [6] (7) resilience of a neuron [40] (8) SDC rate and crash rate [3,27] (9) SDC-1 and SDC-5 where the highest-ranked item predicted by the DNN is not in the top one and top five (respectively) predictions of fault-free execution; SDC-10% and SDC-20%, where confidence score of top-ranked element differs by more than 10% and 20% (respectively) of the fault-free execution. (10) "average mean squared error" of all the network outputs for all the training and testing patterns. This measures how a technique approaches the internal behavior of fault-free or fully-protected execution [55]. (11) Steady-state and peak-temperature of DRAM [35] (12) "Mean executions between failures" (MEBF), which shows the number of correct runs before seeing a failure [47]. (13) For DNNs, "tolerable errors" are defined as those where the CNN output is faulty, but the classification result is unaffected and "critical errors" as those where the classification result is also wrong [4,47,52,53]. Most of these works find that critical errors are much less than the tolerable errors.

(3) *Error-masking behavior of pooling, ReLU and normalization layers:* The max-pooling layer passes only the highest value and ignores remaining values. Thus, faults in the remaining values are automatically masked. However, a fault in the highest value does get passed, which is especially harmful when a fault unduly increases the value of an activation. The ReLU layer ignores negative activations and thus, faults in them are suppressed. The "local response normalization" (LRN) layer masks the faults by normalizing the erroneous values and removing the impact of massive changes. Based on these observations, other error-detectors can be placed after these layers to avoid mitigating the faults that are suppressed by these layers [4].

(4) *Reliability of FC and CONV layers:* Due to full-connection, a fault in an input activation to an FC layer is spread to all the output activations [4]. Further, when the FC layer is not followed by a ReLU layer, a fault in the FC layer is not masked and affects the DNN output directly [57]. Since the CONV layers are followed by pooling or ReLU layers, some faults in CONV layers are masked. However, since the CONV layers are placed towards the beginning of a DNN, a fault in a CONV layer, if unmasked, can affect many layers of the DNN [3,52]. Thus, the overall reliability of CONV and FC layers is determined by the combined influence of the above mentioned factors.

(5) *Reliability of weights and activations:* Activations are more robust than weights [16,18,49,57]. Based on this, activations can be stored using small number of bits than weights [49], and ECC may be used only for weights and not for activations [18].

(6) *Reliability of FP64, FP32, FP16 and FxP:* On performing beam-radiation experiments on FPGA, with decreasing precision (FP64 → FP32 → FP16), there is reduction in the resource utilization and hence, the area exposed to faults [47]. Hence, the FIT rate is reduced. Similarly, on performing beam-radiation experiments on GPU, with decreasing precision there is a reduction in FIT rate and execution time [47]. At the same time, errors in LSBs of an FP16number have higher impact than those in the LSBs of an FP32number [3]. Also, the probability of corrupting the exponent of a number in FP16 format is higher than that in FP32 and FP64 formats [3]. The final resilience of different formats is the combined effect of above factors.
Compared to FxP, FP has larger dynamic range and hence, a fault in the exponent of FP can bring large change in its magnitude. Hence, for the same total bitwidth, FxP format is more robust than FP format [4]. Similarly, for FxP formats with same total bitwidth, the one with less number of integer bits is more robust since a fault in fraction bits has negligible impact but a fault in integer bits can bring a large change in the magnitude [4]. For the same reason, the robustness of a binary-quantized DNN is the highest [15].

(7) *Impact of dataset on reliability:* For a dataset with high number of classes (e.g., ImageNet which has 1000 classes), an error in the top-ranked elements can change the ranking to outside of the top-five class-labels. By comparison, in a dataset with low number of classes (e.g., CIFAR-10 which has only 10 classes), even after occurrence of an error, the ranking may still remain within the top-five class-labels [4].

(8) *Impact of DNN operation and output-format on reliability:* Table 2 classifies the evaluated DNNs based on the purpose for which they are used, viz., image classification, object detection and image understanding. Of these, image classification is the easiest task whereas image understanding is the most challenging task. Clearly, the more challenging a task is, more severe is the impact of a fault in the DNN used for that task. Further, for object-detection, an algorithm which represents the object with vertex coordinates (e.g., Faster RCNN) is more robust than one that represents the object in terms of the coordinates of its center, width, and height (e.g., YOLO) [3] (refer Section 3.5 for explanation). Further, the crash rate of a DNN depends on factors such as CPU-GPU context switch, number of synchronizations, etc. Hence, a DNN that performs more synchronizations has higher crash frequency [3].

3.2. Resilience study of DNN models, layers and design strategies

Jiao et al. [36] study TEs in adder and multiplier units caused due to dynamic variation in voltage and temperature. They assume that when a TE happens, the computation unit returns a random value. They evaluate the accuracy of an MLP and a CNN for different TE-rates. To emulate a TE-rate of, say 20% in the multiplier, they inject errors to 20% of multiplications. The same approach is also used for emulating TEs in the adder. They test with injecting faults in only the adder, only the multiplier and both of them. They note that a fault in the adder has an equal or larger impact on accuracy than a fault in the multiplier. Further, the accuracy on injecting faults in both adder and multiplier is the same as that on injecting faults in only the adder. Hence, faults in the adder have a higher impact on accuracy. This is because when the partial sums are fed to the "activation function", errors from multipliers are averaged, but those from the adders directly affect the input to the "activation function". Clearly, improving the resilience of adder is very important. At the high values of TE-rate, accuracy drops drastically.

They further study the impact of dynamic variations in voltage and temperature on CNN/MLP accuracy. On fixing one of these and varying the other, the accuracy continues to decrease, and thus, both voltage and temperature affect the inference accuracy. Within a specific TE-rate range, compared to CNN, MLP shows a smaller reduction in accuracy, vis-a-vis its baseline accuracy. One reason behind this is that for a given TE-rate, CNN has a higher number of errors due to its more substantial number of arithmetic computations.

Arechiga et al. [17] evaluate the sensitivity of MLPs and CNNs to bit-faults in their weights. The faults are inserted in randomly selected weights after the training is completed. For this, a randomly chosen bit of a weight is inverted. They experiment with varying number of incorrect weights and see the impact on final accuracy.

They observe that with the increasing number of layers and network size, both MLP and CNN become more robust to the errors. This happens because when the number of faulty weights is the same in different MLPs, the fraction of incorrect weights is smaller in a larger/deeper MLP. As the number of defective weights increases, a fewer number of trials reach the baseline accuracy. An exception is that some networks show bimodal behavior such that some tests show very low accuracy and others show accuracy close to fault-free execution. This bimodal behavior is seen in all the CNNs and the smallest (one-layer) MLP; however, larger MLPs do not show bimodal behavior. Overall, the robustness to weight errors is higher in MLPs than in CNNs. They also test CNNs with 3x3 kernels and 7x7 kernels and evaluate the impact of kernel size on robustness. For CNNs with the same depth, those with larger kernel size show higher accuracy with a rising number of errors. For CNNs with same kernel size but different depths, the correlation is rather weak.

Arechiga et al. [6] further evaluate the robustness of VGG16, ResNet50, and InceptionV3. The total number of parameters in VGG16, ResNet50, and InceptionV3 are 138M, 25.6M, and 23.8M, respectively. The number of incorrect parameters ranged between 10^{-7}% to 10^{-6}% of the total parameters. They inject faults in the weights and bias of all three DNNs and in the BN parameters of ResNet50 and InceptionV3. They find that ResNet50 and InceptionV3 show higher robustness to error than VGG16; in other words, they show accuracy loss at a higher percentage of defective parameters. Although the percentage values differ between them, the absolute number of incorrect parameters for a CNN to become unusable is similar in them. Specifically, for VGG16, ResNet50, and InceptionV3, the number of simultaneous bit flips required for making them unstable is 42, 51, and 48, respectively. Unlike VGG16, both ResNet50 and InceptionV3 use BN, which decreases covariance shift and leads to normalization of input to every layer. Hence, the use of BN may increase their robustness towards bit-flips. Further, both CNNs use "shortcut connections" which increase robustness by supplying non-faulty values to the deeper layers.

Hong et al. [2] study the vulnerability of DNN parameters to single-bit flips. A parameter is said to be vulnerable if flipping at least one of its bit can cause a substantial loss in DNN accuracy. They study the criticality of different bit positions for LeNet5 (on MNIST dataset) and AlexNet (on CIFAR10). They find that a change in mantissa bit generally does not lead to large error. A change in 30th bit[2] of an FP32 parameter leads to large error. Flipping of 29th to 27th bits has nearly no impact since their value is one in most DNN parameters. This is because the parameters of these two DNNs range between $[2^{-15}, 2^{1}]$ (refer Table 1). A change in 26th and 25th bits also leads to a substantial loss of accuracy.

As for the direction of bit-flip, only $0 \rightarrow 1$ flip causes large accuracy loss. A $1 \rightarrow 0$ flip can only reduce the parameter value. Due to the normal distribution of parameters, most parameters are inside $[-1, 1]$ range. Hence, a $1 \rightarrow 0$ flip in exponent bit can reduce the magnitude of a parameter by no more than one. Along similar lines, both $0 \rightarrow 1$ or $1 \rightarrow 0$ flips in sign bit cannot lead to large accuracy loss since they alter the magnitude by no more than two. By comparison, a $0 \rightarrow 1$ flip in exponent bits dramatically increases the value of a parameter. Hence, during inference, unduly high activation produced by the faulty parameter value overrides the remaining activations.

[2] Note that in this paper, the 32 bits of an FP32 value are indexed from 0 (LSB) to 31 (MSB), as shown in Fig. 2(a). This convention is different from the paper by Hong et al. [2] which uses indices 1 (LSB) to 32 (MSB) for 32 bits of an FP32 value.

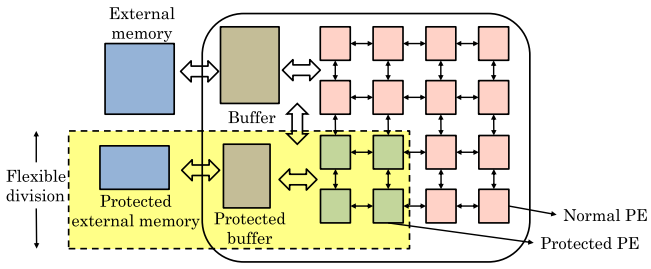


Fig. 5. Using resilience ranking of neurons for adapting the protection strength of a DNN accelerator [40].

They find that nearly 50% of the parameters of the DNNs are vulnerable to single bit-flips. This ratio remains the same, even on doubling the width of DNN. For InceptionV3, the fraction of vulnerable parameters is only 40% due to the presence of auxiliary classifiers. Auxiliary classifiers are used only during training and not during inference, and hence, a fault in them has no impact on the inference accuracy. On removing these auxiliary classifiers, the fraction of vulnerable parameters of InceptionV3 increases to 46%, which is similar to that of other DNNs. They further study how many positive and negative numbers in every layer are vulnerable. Except in the first and the last layer, positive parameters are more vulnerable towards single bit-flips than the negative parameters. This is because ReLU function puts the negative activation values to zero, which are produced from the negative parameters. Hence, a fault in a negative parameter does not pass to subsequent layers. By contrast, in the first and the last layer, both negative and positive parameters are vulnerable. This is because, in the first layer, a fault in a parameter is equivalent to a fault in the input. Also, unlike ReLU, the softmax function used in the last layers does not perform zeroing.

On changing the activation function from ReLU from PReLU (Fig. 4(a) and (b)), even negative parameters become vulnerable, and the fraction of vulnerable parameters increases from $\sim$ 50% to $\sim$ 99%. They also find that the techniques for constraining parameter values such as BN or dropout bring an only marginal reduction in vulnerability.

Schorn et al. [40] present a metric for estimating the error resilience of neurons in DNNs. Intuitively, a change in the output of a neuron that makes a small contribution to the DNN output has less impact on final accuracy. Thus, the resilience of a neuron is correlated with its average contribution to the output such that low contribution implies high error resilience and vice versa. They model the contribution of every neuron to the result of a DNN by using "Taylor decomposition" [64] and "layerwise relevance propagation" algorithm. This algorithm iteratively sends the contribution of every neuron in the $(i+1)$th layer backward to the neurons in the ith layer. The output contribution score of a neuron is averaged over M training images. The mean resilience over all the neurons of a layer is equal to the number of neurons in that layer. This is expected since more the number of neurons in a layer, the less amount of information each neuron contributes. This normalization property allows comparison of resilience between neurons in different layers and hence allows meaningful ranking of the neurons based on their resilience. For example, the neuron with the highest resilience has the lowest priority to receive protection.

They further propose an architecture for utilizing this information, which is illustrated in Fig. 5. Neurons with high resilience are mapped to the normal PEs, which may not be entirely reliable and may give inexact results. The neurons with least resilience are mapped to protected PEs, and their interim and final results are stored in protected on-chip/off-chip memory. The protected hardware components are realized using

ECC or redundancy. Further, the protected computation/storage region can be carefully isolated from the remaining circuit. A limitation of their mapping approach is that it increases routing overhead and reduces flexibility in designing DNN accelerator.

They perform experiments on VGG16 and an "all-convolution neural network". They use two fault models where the output of a fraction of neurons mapped to unreliable PE are modified as follows: (1) setting their output to zero and (2) adding random Gaussian noise to their output. The output of neurons mapped to robust PEs are not changed. They compare their technique with two other techniques which : (T1) assigns resilience value by modifying the backpropagation algorithm and (T2) assigns resilience values randomly. For both the errors models, their technique achieves higher accuracy with increasing fraction of faulty neurons than the other two techniques.

Neggaz et al. [57] note that CONV layers are more resilient to SEs than the FC layers. This happens because the output of CONV layers is masked by subsequent layers such as max-pooling layers. By comparison, any fault in an FC layer that is not masked by a ReLU layer affects the output directly. Since the faults in mantissa have negligible impact, even multi-bit flips in activations do not cause significant errors. Finally, CNNs can easily tolerate faults in PEs; however, faults in memory can lead to a huge drop in accuracy.

3.3. Resilience study of different data-structures and data-types

Reagen et al. [16] present a fault-injection framework for DNNs. Their technique first executes the DNN with FP datatype for all tensors to find baseline accuracy. Then, the DNN is quantized to the minimum FxP width where no accuracy loss is seen compared to the baseline accuracy. The faults are injected either offline or dynamically (i.e., during inference). The overhead of dynamic fault-injection is mitigated by modeling a bit-flip as an element-wise operation and addition of noise as a "linear transformation" of the state. Their technique randomly injects permanent faults in hidden states, activations, and weights. Fault-injection on weights is performed offline. Since hidden states and activations are dynamic values, fault-injection on them is performed during inference. The performance overhead of dynamic fault-injection is below 3.5 ×. On the metric of classification accuracy, their technique is within 12% of a fabricated DNN engine.

Impact of BER on DNN accuracy: They perform experiments using DNNs based on CONV, FC and "gated recurrent unit" (GRU). For all DNNs, the loss in accuracy is negligible up to a small value of BER. For BERs above this threshold, there is an exponential reduction in accuracy. Further, the BER at which different DNNs show 0% accuracy loss varies by orders of magnitude. Thus, the error-mitigation mechanisms need to be selected individually for each DNN.

Impact of quantization: For each DNN, they study fault tolerance of weights with two quantized data types:

- Datatype1: A datatype optimized for each DNN which still provides baseline accuracy. It was found that for all DNNs, this datatype has two integer bits, although the number of fraction bits is different for different DNNs.
- Datatype2: The smallest datatype required for achieving no accuracy loss on all the DNNs. For the tested DNNs, this datatype uses three integer and 13 fractional bits.

They observe that for all DNNs, datatype1 provides higher resilience than dataype2 because the number of integer bits determines the range of representable values and most DNNs show no accuracy loss with only two integer bits. The broader range of numbers in datatype2 leads to faults of larger magnitude, which have a higher impact. Although datatype2 has a higher number of fractional bits, faults in them have a much smaller impact. In other words, the DNN-specific quantization shows high resilience by lowering the number of integer bits from three to two.

Reliability of different DNNs and layers The BER at which a layer shows 0% accuracy-loss differs by several orders of magnitude between layers of a single DNN or between layers of different DNNs. Thus, a layer-specific study of resilience is vital. Further, the number of additional faults required for going from 0% to 10% accuracy-loss is highest for VGG16 and smallest for ResNet50. A smaller value of this number means that the DNN requires stronger protection or wider margins.

Reliability of weights and activations: For most DNNs, the resilience of activations is equal or higher than that of the weights of the same DNN. For example, for VGG16 and ResNet50, the activations are as much as 50 × more robust than the weights. This is because a weight-kernel is multiplied with all the activations of its channel and hence, weight faults impact many values. By comparison, the activations impact a much lower number of values.

Li et al. [4] study fault-tolerance properties of four CNNs, viz., AlexNet, CaffeNet, NiN (all the three trained on ImageNet) and ConvNet (trained on CIFAR-10). CaffeNet has similar properties as AlexNet. They study transient faults in buffers and datapath separately due to the differences in their propagation behavior. They study three FP formats (FP64, FP32 and FP16) and three FxP formats (32b_rb26, 32b_rb10 and 16b_rb10). They define four types of SDCs: SDC-1, SDC-5, SDC-10% and SDC-20%, which are defined in Section 3.1.

Reliability of DNNs and impact of datasets: For the same datatype (e.g., FP32), the SDC rate is highest for ConvNet, since it has only 3 layers and no normalization layer, and hence, it does not perform any error-masking. NiN has second-highest SDC rate. AlexNet, CaffeNet, and NiN work on ImageNet, and for a given CNN and datatype, they have little difference in their four types of SDCs. This is because ImageNet has 1000 classes and on an error in the top-ranked result, the confidence score may change by more than 20%, and the new ranking may be outside of the top five items. By contrast, in ConvNet, four SDC rates are very different. This happens because ConvNet works on CIFAR dataset with 10 classes, and when there is an error in the top-ranked output, it may still remain within the top five items. Hence, the SDC-5 rate of ConvNet is very low. However, its SDC-10% and SDC-20% rates are higher than those of other CNNs, due to the higher sensitivity of confidence values due to the smaller number of classes.

Impact of datatype: For the same CNN, SDC rates vary a lot depending on data types, e.g., for AlexNet, SDC-1 is 0.38% and 7.19% on using FP32 and 32b_rb10, respectively. For FP data types, SDC is caused by a fault in only exponent bits and not sign or mantissa bits. Also, a $0 \rightarrow 1$ bit-flip in exponent has higher chances of causing SDC than a $1 \rightarrow 0$ bit-flip. The reason behind this is that correct values are close to 0, and hence, small differences in mantissa or sign bits have negligible impact. For the same reason, the per-bit SDC rate of FP16 is lower than that of FP32. A fault in the exponent of FP32 causes more significant changes from 0, which has higher chances of leading to an SDC.

In 32b_rb26 and 32b_rb10 datatypes, only integer portions are susceptible. The per-bit SDC rate of 32b_rb26 is much lower than that of 32b_rb10. The reason is that the dynamic range of 32b_rb10 is higher than that of 32b_rb26. Due to this, a faulty value in the 32b_rb26 has higher chances of being closer to 0 than the 32b_rb10. They further note that a fault causing a large change in the numeric value of an activation has a high likelihood of causing an SDC. However, very few faults causing large changes lead to the benign output. When a value is changed significantly, it may not again converge to its original value, which is generally close to 0.

For all CNNs, the activations of a layer lie within a small range. For FP16, FP32, FP64 and 32b_rb10, most faulty values leading to SDCs lie outside this range, but only a few values leading to benign results lie outside this range. The reason behind this is that the dynamic range of these datatypes is larger than that required by the CNNs. The additional range allows higher deflection in value under faults and hence, creates a more significant scope of SDCs. By comparison, 32b_rb26 and 16b_rb10 restrain the maximum dynamic ranges.

Reliability of individual layers: In AlexNet, the SDC rate of first and second layers are much lower than that of remaining layers because the "local response normalization" (LRN) layers placed after first and second layers normalize the faulty values and alleviate the impact of massive changes. Since ConvNet and NiN have no normalization layer, their SDC rates of different CONV layers are almost the same. The SDC rate of layers after "local response normalization" layers are in increasing order since the dynamic range of their values is much narrower than that provided by the datatype. The SDC rates of FC layers are high since a fault in them has a direct impact on the final accuracy, and fully-connected design spreads a fault to all the nodes in the next layer.

Fault propagation and masking behavior: The probability of fault propagation decreases across layers. Although many faults reach the last layer, only a few of them impact the classification accuracy because the "magnitude of change" in faulty values has a higher impact on the SDC rate than the "number" of faulty values. Pooling and ReLU layers mask most of the faults and prevent them from reaching the last layer. Hence, fault-mitigation techniques based on detecting bit-wise differences identify those faults which are eventually masked.

Buffer vs. datapath faults: They observe that the trends in buffer faults are similar to those in datapath faults, although the buffer FIT rates are orders of magnitude higher since (1) high reuse of buffer values spreads the faults (2) the buffer size is much larger than the total number of latches in the datapath.

Salami et al. [51] study fault-resilience of an FC NN with 6 layers using an RTL-level NN accelerator. They study both transient and permanent (stuck-at-0 and stuck-at-1) faults. Since permanent faults persist for the entire duration of inference, whereas transient faults last for a single cycle, permanent faults lead to higher errors than transient faults. Further, most of the bits of input, weight and intermediate registers are zero. Hence, stuck-at-1 faults lead to higher errors than stuck-at-0 faults.

Reliability of different registers: Input registers are most robust to faults since they store only fraction values. Since weight values are multiplied with input values and input values are small, a fault in a weight register dampened. Hence, faults in the weight register are not propagated. By comparison, intermediate registers have the highest susceptibility to faults. This is because they store the longest digit component, and they are used in the adder tree to store the results of multipliers. Hence, a fault in an intermediate register propagates to the next level of adder tree without getting dampened.

Reliability of different layers: A permanent fault in later (near-output) layer leads to relatively lower inference error since compared to the layer sizes, relatively fewer cycles are spent in later layers. By comparison, transient faults in later layers high impact since these faults are not masked by activation functions, and the sizes of NN layers have no impact on transient faults which occur only for a single cycle.

Impact of activation functions and number of PEs: They further study satlin and logsig activation functions (refer Fig. 4(c)-(d)) and find that satlin function leads to higher error than logsig function. Since the NN data is close-to-zero, and the slope of logsig is smaller for small positive input values, it more effectively stops the faults from propagating. With the increasing number of PEs in the accelerator, the inference latency is reduced. Hence, in presence of permanent faults, the erroneous data is read less number of times and the classification errors are reduced. By contrast, a change in the number of PEs does not impact the error-rate in the presence of transient faults since transient faults persist for just one cycle.

3.4. Resilience study of pruning and quantization strategies

Sabbagh et al. [15] evaluate the impact of pruning and quantization on the fault-resilience of two DNNs: LeNet5 (on MNIST) and VGG16 (on CIFAR-10). They evaluate five variants of both these DNNs: original (FP uncompressed), unstructured pruned, structured pruned, binary-quantized, structured + pruned (structured pruning + K-bit quantized where K=3 for LeNet5 and K=5 for VGG16). Refer Section 2.1 for the definition of unstructured and structured pruning. They inject faults only in weights and not in activations or biases.

The accuracy is quantified as the highest BER at which the accuracy loss remains zero. (BERALZ). The higher the BERALZ value, the more resilient a model is. They find that the BERALZ value is highest for binary-quantized and second-highest for structured + pruned. The BERALZ value of unstructured and structured pruned models are the same. The original (baseline) model is the least resilient.

They further study the resilience of different layer types. They first inject faults in every CONV layer. At different BERs, the accuracy of quantized models (where all the CONV layers are quantized) is more stable compared to the accuracy of pruned models. Thus, removal of redundant bits in the weight of CONV layers improves the resilience of the models. When BER is 10^{-3} or 10^{-2}, all models except binary quantized show loss in accuracy. The trends for LeNet5 are similar to those for VGG16, except that at all BERs, the average accuracy loss for VGG16 is higher than that for LeNet-5. Thus, VGG16 has lower error tolerance, which is because it is doing image classification on a more complicated dataset.

On injecting the faults in every FC layer, they find that for LeNet5, the quantized models (i.e., binary and structured + quantized) show a much lower loss in accuracy than the un-quantized models. For VGG16, the binary model shows a much smaller loss in accuracy than the non-quantized models. In the unstructured pruned model, the FC layers are pruned, whereas, in the structured model, FC layers are not pruned. The loss in accuracy in the unstructured pruned model is similar to that in the non-pruned model but is much higher than that in the binary model. Thus, quantization of weights brings greater improvement in resilience than pruning of weights.

Table 5 shows the sparsity of a pruned model, which is the ratio of pruned weights to the total weights. They note that for a pruned model, it is the sparsity and not pruning strategy (structured or unstructured), which decides its fault resiliency. The fault resilience of structured and unstructured pruned models are different for LeNet5 but similar for VGG16. The reason behind this is clear from Table 5. For LeNet-5, on injecting faults in only CONV layers, the resilience of structured is higher than that of unstructured, and on injecting faults in only FC layers, the resilience of unstructured is higher. For VGG16, the resilience of structured and unstructured are similar due to similar sparsity of their CONV layers. For FC layers, the sparsity values differ between structured and unstructured models but are still small, and hence, its impact on resilience is also small.

For designing resilient DNNs, they propose the following guidelines: (1) binary quantization should be used for all layers since not quantizing few layers leaves them vulnerable to faults. (2) Structured pruning of CONV layers and unstructured pruning of FC layers with high sparsity improve resilience and also allow efficient computations on multi/many-core systems.

3.5. Resilience studies on GPUs and FPGAs

Since GPUs have been traditionally used for error-tolerant graphics applications [65], they do not use advanced schemes for improving reliability such as those used in CPUs. In GPUs, only SECDED ECC is used, which protects only the prominent components such as instruction and data caches, RF and main memory [20]. GPUs have a large number of flip-flops in pipeline registers and internal queues which are not protected. Similarly, the warp scheduler and some control/ computation logic resources are not protected. A fault in these components can escape the ECC protection [27]. Further, a fault in shared memory or scheduler can corrupt several parallel threads or kernels. Also, the value of a single faulty thread may be consumed by many future threads, leading to multiple errors.

Lunardi et al. [27] study the reliability characteristics of various applications running on GPUs. They perform experiments using two GPUs: K40 and Titan X and their configuration is shown in Table 6. The

Table 5
Sparsity of overall DNN, CONV layers and FC layers for LeNet and VGG16 [15].

	Total sparsity		Sparsity of CONV layers		Sparsity of FC layers	
	LeNet5	VGG16	LeNet5	VGG16	LeNet5	VGG16
unstructured pruning	95.6%	93.6%	65%	89.2%	94.6%	46.7%
Structured pruning	5.8%	94.3%	93.8%	93.8%	0%	0%

Table 6
Key characteristics of K40 and Titan X GPUs [27].

	Architecture	Process node	Transistor technology	Reliability scheme
K40	Kepler	28 nm	CMOS	It uses SECDED ECC for RF, caches, and shared memory and parity for read-only data cache
Titan X	Pascal	16 nm	FinFET	No ECC is used

differences in transistor technology affect the rate of bit-errors and the differences in microarchitecture impact how bit-level faults manifest as application error.

They evaluate many workloads, including YOLO, a DNN-based object detection algorithm. They perform both neutron beam radiation and software-level fault-injection. Software-level fault injection is performed (1) at the instruction output to find PVF and (2) in the RF to find the RF AVF. This allows comparing the impact of faults in unprotected computing resources and in ECC-protected memories. They choose input dimensions such that the utilization of processor components in the irradiated area becomes high.

Impact of ECC: They find that ECC decreases the SDC FIT rate for all the workloads by nearly 10 ×. For YOLO workload, ECC does not reduce those SDCs that affect the accuracy of object detection. ECC does reduce other SDCs seen in YOLO, including the inconsequential SDCs. In other words, ECC cannot reduce most severe errors, which makes it ineffective for accuracy-demanding applications such as autonomous driving. Since SECDED ECC causes a program to crash on detecting a double-bit error, it increases the crash rate of K40. However, when ECC is disabled, the same double-bit error may cause an SDC.

Impact of FinFET: For all workloads, Titan X shows lower SDC FIT rate than K40, and this confirms that FinFET is more robust than CMOS. In fact, for compute-intensive workloads such as YOLO and GEMM, the FIT rates of Titan X are closer to that of "K40 with ECC". Thus, FinFET is even more effective in mitigating faults in logic than in memory. Most radiation-induced faults impact multiple dimensions and values of the result.

Comparison of AVF and PVF: They use SASSIFI [66], an architecture-level tool for fault injection in GPU by NVIDIA. Using profiling, SASSIFI ascertains all registers and instructions where a transient fault can be injected. By injecting faults in instructions, the vulnerable instructions can be identified and thus, the PVF of the code can be measured. By injecting faults in the registers, AVF can be measured. For all the benchmarks, AVF is lower than the PVF, and hence, faults affecting memory are less likely to propagate to the final result than the faults occurring during the computation. This is because incorrect memory locations may not get consumed or may get overwritten and the applications running on GPU generally do not have high data reuse. By comparison, the output of an instruction is likely to be consumed in future computations. Finally, they note that radiation-induced events in unprotected compute elements have a significant impact on the computations. Overall, they show that faults in unprotected GPU components have a higher likelihood of impacting final accuracy than the faults occurring in the protected memory components.

Fernandes dos Santos et al. [3] evaluate three DNNs viz., YOLO, Faster RCNN, and ResNet on Jetson TX1, K40 and Titan X GPUs. *DNN characteristics:* YOLO and Faster RCNN perform object detection using two FC layers. ResNet performs object classification using one FC layer. ResNet is much deeper than YOLO. Since YOLO runs fewer operations and has less complexity, its SDC rate is likely to be smaller than that of Faster RCNN. However, the deep pipeline of ResNet and object representation of Faster RCNN bring down the number of critical SDCs. A critical SDC is one that changes the coordinates of a bounding-box (in YOLO and Faster RCNN) or changes the ranking of the object (in three DNNs).

Faster RCNN works in two stages such that errors affecting a region not evaluated by region-of-interest (RoI) pooling are masked. Also, in Faster RCNN, objects are composed based on multiple anchor boxes and anchor scores. An error in an RoI that changes the anchor boxes is still a tolerable error. By contrast, in YOLO, detection is performed by FC layers, and hence, the criticality of an SDC is determined by how it impacts the output of the last CONV layer. In YOLO and Faster RCNN, only 1% of the SDCs that are critical for classification are not critical for detection. This is because any error in CONV data impacts both detection and classification.

Different types of errors and their behavior: Based on their distribution, they classify errors as: (1) single (2) line (multiple elements on a single row/column of an output matrix become incorrect) (3) rectangle (at least four elements in a square/rectangle become incorrect) (4) cubic (multiple values in three dimensions of the output matrices become incorrect) and (5) randomly distributed errors.

They note that rectangle errors have the highest impact on DNN reliability. In GEMM, rectangle errors arise due to faults impacting the scheduling or execution of several threads of a streaming multiprocessor (SM). If a fault causes a thread to be wrongly mapped or scheduled on an SM or if thread-synchronization fails, the entire SM output matrix region is corrupted, which leads to a "rectangular error". Faults in ECC-protected memories such as RF and caches result in single or line errors. ECC corrects the less-critical errors viz., single and line errors but not the remaining errors. Hence, on using ECC, the fraction of rectangular errors are increased.

Comparison of FIT rate, SDC rate, and crash rate: They note that the FIT rate of ResNet is higher than that of YOLO and is similar to that in Faster RCNN. ResNet has a vast number of layers which increase its error rate that cannot be compensated by its simpler CONV approach. For GEMM computations, SDC rate is higher than the crash rate. Except for Faster RCNN on Titan X, the probability of crashes is higher than that of SDCs. Although DNNs have an inherent capability to mask SDCs, they cannot mask crashes since crashes depend on other factors such as CPU-GPU context change and GPU control logic. For object-detection operations, multiple host-device synchronizations are required, and a soft error during these synchronizations can lead to a crash. Since Faster RCNN needs many more synchronizations, its crash frequency is an order of magnitude higher than that of YOLO.

Results on TX1: Jetson TX1 has an integrated GPU on the same chip as the GPU [67]. On performing beam radiation, the crash rate of TX1 is higher than that of K40, and Titan X since the embedded CPU of TX1 is also irradiated whereas the CPU of K40 and Titan X were kept far away and hence, were not irradiated. Further, Faster RCNN has high latency on TX1 and hence, data stay in caches and RF for a long time and remain vulnerable to faults. This results in higher SDC rate of Faster RCNN on TX1. Also, since TX1 uses the same hardware for performing

multiple computations for a single layer, a critical SDC severely impacts its detection accuracy.

Effect of mixed/low-precision on reliability: On Titan X, Faster RCNN uses mixed-precision libraries which increase the number of concurrently executed instructions. An error in a 16b value or operation has a higher likelihood of propagation than an error in a 32b value. Similarly, errors in LSBs of a 16b number have higher severity than those in the LSBs of a 32b number. The chances of corrupting the exponent of a number in 16b format are also higher than that in 32b format. Hence, the use of mixed-precision increases the FIT rate. For example, YOLO processes 20X more frames on Titan X than it does on K40 (with no ECC) whereas Faster RCNN processes only 20% more frames on Titan X than it does on K40.

Comparison of critical SDCs and total SDCs: They further note that the number of critical SDCs is much less than the total SDCs because (1) not the entire output is used for detection (2) despite the fault, the shape of the object may still be recognizable, which allows correct detection (3) a change in detection probability may not change the ranking. (4) On conversion of object coordinates from FP into integers, faults in lower bits may not affect the detection accuracy.

Faster RCNN has a much lower fraction of critical SDCs than YOLO. This is because Faster RCNN uses multiple anchor boxes for defining an object and thus, an error in one of the coordinates of a vertex does not alter the shape. By comparison, YOLO represents an object in terms of the coordinates of its center, width, and height. Corruption of center-coordinates shifts the object location, and thus, the actual object is not detected, reducing both recall and precision. Thus, to achieve higher reliability, it is better to represent the object with vertex coordinates.

Impact of ECC: Although ECC decreases the SDC rate of GEMM by more than 10X, ECC has lower effectiveness on a DNN since some errors removed by ECC would not impact DNN output. Thus, ECC does not lower the number of "critical SDCs" for three DNNs. ECC increases the fraction of rectangle errors, which are not masked and corrupt the final results. Also, on using ECC, the average difference between the faulty and correct value becomes larger. Due to this, faults not corrected by ECC have a high impact on DNN output.

They inject multiple types of faults such as random/zero value, single bit-flips and multiple bit-flips. Their fault-injection studies show that the faults injected at the input of the two FC layers do not lead to SDCs. Since FC layers appear only at the end of YOLO, faults in inputs to them do not become cubic errors. Hence, these faults are suppressed due to the inherent resilience of FC layers. Rather, the faults in CONV layers are primarily responsible for the SDCs at the CNN output. Max-pooling layers mask SDCs by virtue of discarding most of their inputs. Specifically, they mask single and random errors. The line errors occurring in only the first layer are suppressed. The cubic and rectangular errors are not suppressed. Thus, the errors masked by max-pooling layers are the same (single and line) which are also masked by ECC.

Since YOLO performs 3D CONV on 3D matrices, even a single error impacts 3D area. Thus, while propagating to subsequent layers, a single error becomes a cubic error which cannot be corrected by max-pooling layer. Due to this, with increasing layer-index, an increasingly larger fraction of data values are impacted by the error.

Fernandes dos Santos et al. [47] study the impact of storage and compute precision on the reliability of FPGA (Zynq-7000) and GPU (Titan V). They evaluate several traditional applications, including GEMM, in various precision. In addition, they evaluate MNIST CNN on FPGA and YOLOv3 CNN on GPU. They train with single-precision and use the same weights for half and double precision. This allows isolating the impact of mixed-precision execution. They perform both neutron beam and fault-injection studies.

Metrics: To study the interplay between reliability and performance, they also report "mean executions between failures" (MEBF), which shows the number of correct executions finished before an error is seen. A higher value of MEBF is desirable. Also, they define "acceptable relative error" (ACRE) as the maximum difference from the golden result that is still considered as acceptable.

Results on FPGA: A fault in configuration memory of FPGA corrupts the implemented circuit, and the memory remains corrupted until a new bitstream is loaded into it. Once the circuit is corrupted, the output continues to be wrong and hence, they reprogram the FPGA after each observed error. On FPGA, with decreasing precision, the circuit size (area) decreases, which decreases the error rate. Although MNIST CNN requires higher resources than GEMM, it has lower FIT rate due to the fault-tolerant nature of CNN.

On increasing ACRE from 0% to 0.1%, FP64 circuit sees a large reduction in FIT since 63% errors can be tolerated. However, for the same change in ACRE, the reduction in FIT is much smaller for FP32 and is negligible for FP16. The reason behind this is that with decreasing precision, a fault is more likely to corrupt the MSB of a data and hence, lead to an error with larger magnitude. They find that for MNIST CNN, critical errors are much less numerous than tolerable errors. However, critical errors increase on going from double to half-precision. For both GEMM and MNIST CNN, MEBF is increased on going from double to half-precision.

Results on GPU: On GPU, the correlation between precision and reliability is not straightforward due to the impact of architecture and compiler. Volta GPU has custom hardware for executing mixed-precision computations: 2688 cores for FP64 and 5376 cores for FP32/FP16 computations. They find that the number of used registers are the same for FP32 and FP16 operations but increase to 2X for FP64 operations. Since the device FIT rate depends on the actively used hardware resources, fewer cores would get corrupted during FP64 execution. However, FP64 cores are larger and more complex. Also, the number of exposed bits in RF and memory are 4X for FP64 than for FP16. In general, arithmetic operations take 8 cycles for FP64, 4 cycles for FP32 and 6 cycles for FP16 (two FP16 operations are concurrently executed).

To get more insights, they first execute addition (ADD), multiplication (MUL), and fused-multiply-add (FMA) instructions in different precisions. For MUL, the FIT rate is higher for higher precision due to higher complexity and wider registers. For ADD, the trend is opposite due to simpler hardware and a larger number of active cores for FP16/FP32 than FP64. The trend for FMA is the combined effect of trends for MUL and ADD. The decreasing order of FIT rates are FMA, MUL and ADD, which is because (1) FMA circuit has higher complexity and (2) the execution latency depends only on the data precision and not on the instruction type (FMA, MUL or ADD).

For YOLOv3 CNN, the FIT rate decreases on going from FP64 to FP16. For FMA and GEMM, the DUE rate for FP64 is 2X that of FP16 due to the higher execution time with FP64. This issue is more severe for YOLOv3 CNN since object detection CNNs have a higher likelihood of experiencing DUEs than arithmetic operations. The reduction in FIT rate as a function of ACRE is similar for FP32 and FP16, whereas the reduction is higher for FP64. Further, similar to the case of MNIST CNN on FPGA, FP16, and FP32 have a higher fraction of critical errors than FP64.

For all benchmarks (GEMM, YOLOv3, ADD, MUL, FMA), MEBF increases with decreasing precision due to the reduction in execution time and FIT rate. Although the FIT rate of microbenchmarks (ADD, MUL, FMA) is similar to that of YOLOv3, FP16 and FP32 bring much greater improvement in MEBF for YOLOv3 than for microbenchmarks. This is because FP16 and FP32 save memory bandwidth by reducing memory accesses. Thus, low-precision formats improve *both* performance and reliability and hence, are promising for memory-constrained applications running on GPUs.

4. Techniques for improving resilience

In this section, we begin with an overview of prominent ideas of different works (Section 4.1). We then discuss the algorithm-level

Table 7
A classification of error-mitigation techniques.

Optimizations of DNN architecture	Use of normalization layer [2,4–6], shortcut connections [6], increasing the number of neurons [46,63]
Anomalous value detection	[3,3,4,37,49,68]
Replication (e.g. TMR)	[10,39,52,55,63,69]
Re-execution	[10,37]
Use of Razor flip-flop	[14,42,48,49]
Criticality-aware protection	Mapping more resilient neurons to less reliable hardware [40,42], storing high-order bits of data in DRAM regions with normal refresh rate [18,41], storing more critical bits in resilient dies of 3D DRAM [25], using TMR only in the most vulnerable layers of a NN [52], protecting only activation function and adders in the output layer [8]
ECC	[3,18,27,40]
Using SE-resilient cell-design (hardening)	[4,27,39]
Masking/pruning schemes	bit masking [25,49,51], word masking [49,51], fault-aware pruning [12]
Fault-aware training	[5,9–11,44,50]
Others	skipping ineffectual computations [14,49], reducing memory accesses using tiling and scheduling [35], data-encoding [70], spatial-unfolding [8]

(Section 4.2) and architecture-level (Section 4.3) techniques for error-mitigation. Further, we discuss techniques for addressing errors in DRAM memory (Section 4.4) and those arising due to voltage-scaling (Section 4.5). Finally, we review works that provide IP protection by leveraging reliability issues and show their implications on security (Section 4.6).

4.1. Overview and key ideas

Table 7 presents an overview of error-mitigation techniques.

We now summarize some of the major observations and key ideas of different resilience techniques.

1) *Criticality-aware protection:*
   - Different components of a DNN have different criticality. For example, in a layer with a small number of neurons, each neuron makes a much higher contribution to output and hence, is more critical [40,63]. Similarly, a fault in the mantissa bits has a negligible impact on the overall accuracy [2], whereas a fault in certain exponent bits has a high impact on accuracy [2,18,25].
   - Several techniques leverage this observation to reduce the overhead of protection schemes and increase their effectiveness. At DNN algorithm-level, the impact of highly critical neurons can be reduced by changing the value of weights [46,63] or the placement of layers in the DNN [46].
   - At accelerator-architecture level, components with higher criticality are stored in regions with stronger protection or less error-rate. For example, filters with high sensitivity weights are mapped to the column of MAC array having lower mean values of TE rates [42].
   - Other ideas include mapping more resilient neurons to less reliable hardware [40], storing high-order bits of data in DRAM regions with normal refresh rate and low-order bits in DRAM regions with reduced refresh rate [18,41], storing more critical bits in resilient dies of 3D DRAM [25], hardening only MSBs of FxP data [4], storing MSBs in (robust) 8T SRAM and not (vulnerable) 6T SRAM [59], using TMR only in the most vulnerable layers of a NN [52] and protecting only activation function and adders in the output layer [8], etc. The limitations of these techniques is that they require non-trivial changes in the accelerator architecture.
2) *Fault-aware training:*
   - Some works account for the fault-pattern of the inference accelerator during training itself [5,9–11,44,50]. Some works store both FP and FxP weights [9,50], or both inexact and exact weights [5] to ensure that the training process converges. Others perform BWP in the digital domain and FWP in analog domain [11].
   - Some works perform retraining to reduce the impact of errors [8–12,18]. The overhead of complete retraining is avoided by retraining only the fault-prone FC layers [11].
3) *Layer-specific optimizations:* Some works apply different optimizations to different layers such as different voltage scaling ratio [14,19] and batch size [35] for different layers and saving weights of different layers with different bitwidths [11,49].
4) *Limitations of ECC:* ECC has several limitations in the context of DNNs used in image-processing workloads. ECC mitigates those errors that are mitigated by max-pooling layers also [3]. ECC cannot prevent rectangle and cubic errors which have the highest impact on classification accuracy [3]. SECDED ECC, which is generally used in product systems, is ineffective in protecting against double-bit error [27]. In GPUs, ECC is used only for memory-elements, but the faults in remaining unprotected components have a higher probability of impacting final accuracy than the faults occurring in the ECC-protected memory components [27]. In fact, use of SE-resilient transistor technologies such as FinFET is more effective than use of ECC [27].
5) *Value-range based optimizations:* Several optimizations are based on the observation that the magnitude of faulty-free weights are generally small.
   - Bit-masking [25,49,51], word-masking [49,51] and fault-aware pruning [12] techniques reduce the magnitude of a faulty weight since a fault generally increases the value of a weight, and the correct value of a weight is small.
   - Since the fault-free values are small, the activation function that performs more aggressive thresholding is also more effective in stopping the faults from propagating [51]. Similarly, compared to FP datatype, use of FxP datatype with a small number of integer bits and binary datatype reduces the error-rate by clipping the magnitude of faults [4,15].
   - In a systolic-array based DNN accelerator, each MAC makes an only small contribution to the output of a neuron and hence, on a TE in a MAC, the update from its next MAC can be dropped to allow the faulty MAC to correctly finish its update [14].
   - Since the correct values of weights are close to 0 and a $0 \rightarrow 1$ flip increases the magnitude of the weight, unduly high activation produced by the faulty weight value during inference, overrides the remaining activations. Due to this, a $0 \rightarrow 1$ flip has a much higher impact on accuracy than a $1 \rightarrow 0$ flip. For the same reason, a stuck-at-1 fault is more harmful than a stuck-at-0 fault. Based on this observation, a special cell can be designed that consumes low leakage energy when it stores zero value [39] and completely avoids $0 \rightarrow 1$ bit-flip. Similarly, data-encoding techniques can be used to alter the distribution of 0s and 1s [71].

Table 8
The configuration of MNIST CNN [46].

Layer	Filter shape	Output shape
0	3x3	16x28x28
1	3x3, max pooling 2x2	64x14x14
2	3x3	64x14x14
3	Global average pooling	64x1x1
4	1x1	10x1x1

- The typical values of weights/activations of a DNN are in a fixed range. By recording this range in the fault-free execution, during faulty execution, incorrect values can be identified as those that lie above this range [3,4,37,68]. Also, if the range in the fault-free execution is such that some bits of FP32 representation are same for all the weights, then ECC information can be stored in those weights [18].

6) *Mitigating errors due to low-voltage operation:* The strategies for dealing with errors due to low-voltage operation (such as TEs) include increasing the voltage on detecting a TE [48], ignoring the multiplication which shows a TE and forwarding the unmodified partial sum [12,42], identifying input sequences that lead to TEs and in future, boosting the voltage on detecting these sequences [48], using failure-prone cells as in-situ canary circuit and boosting the voltage of remaining crucial cells when these canary cells fail [50], and boosting the voltage of SRAM on read/write operation [19]. The voltage of a MAC unit storing a zero weight need not be boosted [48].

4.2. DNN algorithm-level techniques

Schorn et al. [46] present a technique for improving the resilience of DNNs. They note that CONV layers extract features at different locations of their input by sliding multiple "weight kernels" over the input. This leads to a "feature map" for every kernel. They obtain global relevance score of every feature by adding together the relevance values of all neurons over the spatial dimensions. Further, although a fault in the output of a layer affects only one value in a feature map of that layer; when passed to the next layer, it spreads to all the feature maps. They compute the impact of the initial error on the features with different "relevance values" in the next layer. Although the propagation of error depends on the nature of subsequent non-linear layers, they define a "feature bit-flip criticality score" based on the amplitude of the weights connected to the erroneous neuron. A high value of this score implies that a fault in a feature of the previous layer has a high impact on those features in this layer whose average relevancy value is high.

To avoid the need for extra protection of critical portions in DNN, they seek to distribute the resilience in a DNN uniformly. With this objective, they propose two schemes: (1) changing layer placement and (2) changing the weights.

(1) Since layers with very few neuron outputs show low resilience, they suggest minor modifications to DNN architecture for avoiding such vulnerable layers. For example, as shown in Table 8, the CNN used for the MNIST database has only 64 output neurons in the second last layer. Hence, this layer has a higher vulnerability than the other layers. To reduce the vulnerability of this layer, their technique moves the "global mean pooling layers" at the end and places the 1x1 CONV before them. This change increases the output size of the second-last layer from 64x1x1 to 64x14x14. This change has no impact on the number of weights and negligible impact on the CNN accuracy, yet it significantly reduces the worst-case failure rate on a bit-error. The limitation of this scheme is that for larger CNNs, such as those trained on ImageNet dataset, this scheme exacerbates the number of operations and weights.

(2) To equalize the "criticality scores" of different features in a layer, they propose adapting the weights of the next layer. Specifically, weights connected to a feature with more than average "criticality score" are decreased, and weights connected to a feature with less than average "criticality score" are increased. The loss in accuracy is recovered by performing a few additional training iterations. This scheme does not change CNN architecture and does not increase the number of computations. This scheme is especially useful in reducing the failure rate of large CNNs.

Eldridge et al. [63] propose using k-modular redundancy in NN. Starting from a non-redundant NN, a k-modular NN is created by replicating the hidden neurons by k-fold and dividing their weights by k times. Thus, unlike in the conventional redundancy scheme, voting is not performed. They evaluate their technique on an NN accelerator and experiment with three applications: Blackscholes, RSA, and Sobel. With the increasing value of k, the accuracy of Sobel increases greatly, whereas the accuracy of Blackscholes and RSA increases by a small amount and plateaus at $k \sim 3$. This is because Sobel NN is much smaller than the other two NNs and hence, each neuron of Sobel NN makes a much higher contribution towards the application output. However, with increasing value of k, the latency/energy penalty of their technique increases linearly.

Xing et al. [9] propose a technique for adapting the training process to account for the reliability issues in DNN accelerator. They assume that the training is performed on the GPU (or CPU) and the inference is performed on the accelerator. They note that high fault rates due to overclocking or soft-errors degrade the accuracy of a DNN accelerator. They propose a technique to account for these sources of error during DNN training itself. The overall flow of their technique is shown in Fig. 6. In their technique, during training, FWP is performed on the error-prone accelerator and the BWP is performed on the (error-free) GPU. FWP is done in FxP, which reduces resource utilization. BWP is done in FP for ensuring the accumulation of small changes in the parameters. The training is complete when the loss goes below the threshold, and it implies that the trained DNN can be safely used on the error-prone accelerator.

They evaluate their technique on an FPGA-based accelerator. They use four CNNs: LeNet, AlexNet, VGG-16, and VGG-19 for which baseline frequency is 210, 210, 190 and 190 MHz respectively. For AlexNet, VGG-16, and VGG-19, retraining improves accuracy at the extreme overclocking frequency (260MHz for AlexNet and 240MHz for VGG). For LeNet, the accuracy remains unaffected till 260MHz and drops abruptly at 270MHz. Retraining has no impact on the accuracy of LeNet. They also evaluate the impact of soft-errors and find that the trends are similar.

A challenge in their technique is that it requires conversion between FxP and FP values in each iteration of the training. This may decrease the accuracy and increase the number of iterations required for convergence. Further, performing FWP and BWP phases on different platforms leads to a large amount of data transfer, which is especially harmful in the case of large CNNs.

Fernandes dos Santos et al. [3] note that the fraction of GPU processing spent in matrix multiplication is 67%, 82% and 80% in YOLO, Faster RCNN, and ResNet, respectively. Hence, they evaluate an "algorithm-based fault-tolerance" (ABFT) approach for protecting matrix multiplication. The ABFT scheme is implemented only in YOLO since ResNet, and Faster RCNN are implemented with closed-source libraries. To implement the ABFT approach, they replace the "last row" of the first input matrix and the "last column" of the second input matrix with "column checksum" and "row checksum", respectively. Although the ABFT scheme does not bring a large reduction in SDCs or crash rate, it can remove nearly half of the SDCs on K40 and Titan X. The ABFT approach corrects nearly 87% of the critical errors. Thus, the ABFT scheme is more effective than ECC in reducing critical SDCs since it mitigates all the errors affecting GEMM operations. Also, the ABFT scheme incurs lower overhead than ECC.

For designing resilient max-pooling schemes, they note that it is important to detect and correct errors in early layers so that their impact is not magnified. Also, for all the frames of the tested datasets, the maximum input value to any pooling layer in error-free execution is 21. This value is much smaller than the faulty value that radiation can produce.

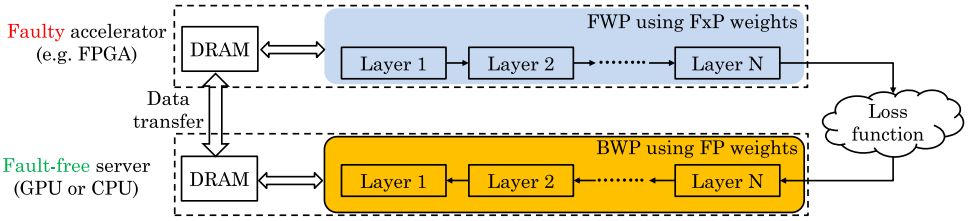


Fig. 6. The modified training scheme of Xing et al. [9] which performs FWP using FxP weights on the accelerator and BWP using FP weights on the GPU.

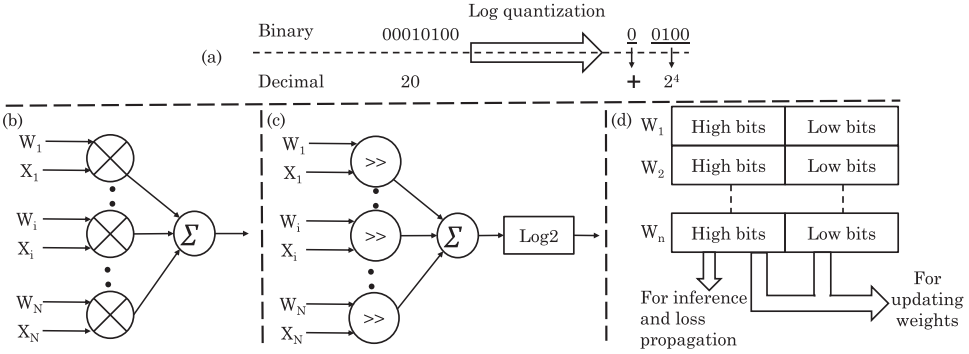


Fig. 7. (a) An example of log quantization [11]. It allows replacing (b) multiplication operation with (c) shift operations. (d) Use of high-order bits for inference and loss propagation, and all the bits for updating the weights.

Based on these observations, they propose two resilient max-pooling schemes: (1) this scheme checks whether the value of the maximum element is higher than a threshold. If so, the present frame is not processed. The threshold can be chosen as 10 times the maximum value of a fault-free execution. This scheme detects faults that corrupt several elements and have the highest impact on the final value. (2) If the maximum element is higher than a threshold, this scheme passes the second maximum element, if the second maximum element is below a threshold. This scheme does not degrade the accuracy much since max-pooling is inherently an approximate operation. Compared to the $O(\log N)$ overhead of ECC, this scheme has an overhead of $O(1)$ only. They show that their pooling scheme correct more than 80% of the SDCs. Also, it can detect 98% of the SDCs under radiation experiments, which is close to the 99% detection requirement imposed on autonomous vehicles.

Jia et al. [11] study the impact of process variation (PV) on analog NN accelerator. They note that the accuracy loss due to PV is higher in deep NNs than in shallow NNs since deep NNs have higher degree of non-linearity and higher amount of error accumulation. Further, due to PV, the operating points of nearby processing units may differ, leading to a mismatch. To mitigate the impact of PV, retraining can be performed, however, training all the parameters of a DNN in analog domain is challenging since convergence may take a lot of time.

They note that CONV layers are more fault-tolerant than FC layers. Based on this observation, they perform transfer learning in the digital domain whereby the parameters of FC layer are adjusted based on the fault-pattern of an individual chip. Thus, instead of training all the layers, only FC layers are retrained. The back-propagation algorithm is implemented using low-precision in digital domain. This ensures that convergence is reached and resource consumption is lowered. Using low-precision analog-to-digital converter (ADC), neuron values are transformed into digital domain. Log quantization is performed on the gradients and then, they are saved in narrow bitwidth. For example, if the original binary value is 0001 0100, then after log quantization, the value is represented as $+2^4$. Here, the MSB determines the sign bit. This is shown in Fig. 7(a).

The loss in accuracy due to log quantization can be tolerated by the NNs. Log quantization increases the range of representable numbers in the fixed storage budget. For example, the range of representable integer numbers with 4b storage is [-8, 7] and after log quantization, the range becomes [-128,128], which is comparable to the range of representable integer numbers with 8b storage ([-128,127]). Thus, log quantization increases the range by sacrificing the precision. Log quantization also allows replacing the multiply operations of back-propagation phase with the shift operation, which improves area and energy efficiency. This is shown in Fig. 7(b)-(c).

Table 9
Characteristics of LeNet-5 and CIFAR-quick.

	LeNet-5	CIFAR-Quick
Bitwidth required for training	10b	18b
Learning Rate	0.01	<0.0001
ADC required for ensuring training convergence	6b	5b

They further note that during training, different CNNs require different bitwidth of data. For example, LeNet-5 and CIFAR-quick require 10 and 18 bit weights. This is because the "learning rate" of LeNet-5 is 0.01 which is more than 100X higher (nearly 7 bit difference) than that of CIFAR-quick. Based on this, they save CONV layer weights with low bitwidth and FC layer weights with higher bitwidth. As shown in Fig. 7(d), the high bitwidth weights are used only for updating the weights in the training stage. During loss propagation and inference phase, only their high-end bits are used.

To balance the overhead and accuracy advantage of ADC, 6b and 5b ADC are used with LeNet-5 and CIFAR-quick, respectively. These observations are summarized in Table 9. Their proposed changes do not affect the inference phase. They show that their technique greatly improves tolerance of operating point drift and mismatch with only minor loss in NN accuracy. Also, compared to performing 32b FP training, their technique reduces memory requirement and also saves energy.

4.3. DNN accelerator architecture-level techniques

Li et al. [4] propose multiple techniques for mitigating faults in DNN accelerators. They propose using datatypes that provide just-sufficient numeric range and precision for values of a layer. For example, using 32b_rb26 instead of 32b_rb10 reduces the FIT rate of datapath by more than 200 ×. On applying this optimization, remaining SDCs arise due to bit-flips in most significant bits of FxP. Hence, they harden only these bits to reduce SDCs. These hardening schemes reduce the FIT rate of datapath by two orders of magnitude with only 20% area penalty. They also suggest using normalization layers in CNNs and placing error-detectors after them to avoid identifying benign faults.

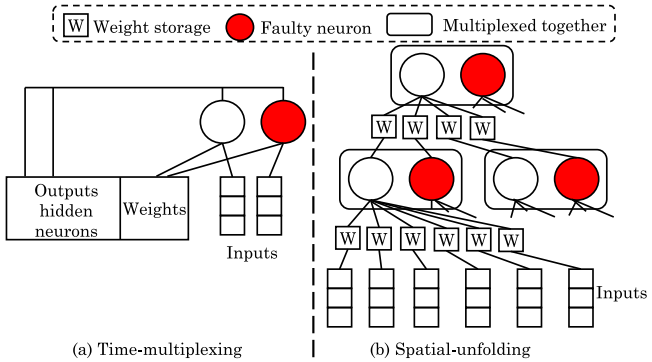


Fig. 8. Implementation of an ANN accelerator using (a) time-multiplexing or (b) spatial-unfolding approach [8].

Although adding on-chip buffers allows exploiting data reuse, they also have a high vulnerability and are difficult to protect through ECC. To protect them, they propose a "symptom-based fault detection" scheme. This scheme first learns the typical range of value in each layer with fault-free execution and adds a 10% guardband to be conservative. Then, during actual execution, the values are compared with these limits to identify any faults. The latency of detection is hidden by performing the check when the output data of one layer is being used as the input during execution of the next layer. This technique is effective only for those networks and datatypes that show strong symptoms, i.e., where the values lie within a reasonable range.

The "ISO 26,262 standard" for the safety of road vehicles prescribes that the DNN inference engine should have an SDC FIT rate below 10 [72]. With no protection, the FIT rate of Eyeriss accelerator is much larger than this limit, but their techniques bring the FIT rate within this limit.

Temam et al. [8] propose an ANN accelerator design for achieving high fault-tolerance. An ANN accelerator can be designed in either a time-multiplexed or a spatially-unfolded manner. These designs are shown in Fig. 8. They assume a "spatially-unfolded" ANN accelerator where each neuron is mapped to hardware and the synapses are stored in storage locations at the level of every neuron. Spatial-unfolding reduces data-movement energy by bringing neurons (compute-units) closer to the synapses (storage-units).

Further, compared to the time-multiplexing scheme, this scheme increases weight-bandwidth and reduces the latency of accessing the weights. In a time-multiplexing design, the multiplexing process itself requires complicated logic such as routing and address decoding. Also, an erroneous transistor can have a high impact on the accuracy of the accelerator. In the spatially-unfolded design, the control logic is used only at the input/output stage and by designing the control logic with more reliable transistors, its vulnerability can be easily reduced. A fault in single or even multiple synapses or neurons have a negligible impact after retraining. Further, if the ANN size exceeds the size of the spatially-expanded accelerator, time-multiplexing can be used to reuse the computation resources.

They study a fully-connected ANN with 90 inputs, 10 hidden neurons, and 10 outputs. They assume 16b fixed-point data values. They randomly inject a different number of faults in a logic operator or latch. They select 10 applications from "UCI machine-learning repository" as benchmarks and study the neural network accuracy after retraining. They observe that for all the benchmarks, up to 12 faults can be tolerated, regardless of which bit shows a fault. Some benchmarks show negligible error even when 20 faults are introduced. This is because a neuron with significant error can be suppressed during retraining. Beyond 20 faults, there is a sharp degradation in accuracy.

In the output layer, a fault in the adder just before the activation function, or in the activation function itself has a large impact on the accuracy. In fact, the larger the magnitude of the change in value caused by the fault, the more likely it is that the predicted class will be changed. Thus, the activation function and adders in the output layer show high sensitivity to faults. By protecting them, the resilience of ANN accelerator can be improved.

Libano et al. [52] study the reliability of two NNs on SRAM-based FPGAs: (1) "Iris Flower ANN", an FC network used for classifying an Iris flower image into one of the three flower subspecies. It has one hidden and one output layer. (2) MNIST CNN which has the following structure: CONV1-POOL1-CONV2-POOL2-FC1-ReLU-FC2. They inject faults in the configuration bits of configurable logic blocks (flip-flops, lookup tables, DSPs and interconnections). No faults are injected in configuration bits of block RAM.

The results from radiation experiments show that for both the networks, the frequency of tolerable errors is much higher than that of critical errors. For Iris Flower ANN, the faults in the hidden layer have a higher probability of generating tolerable and critical errors than the faults in the output layer. This is because faults in the hidden layer affect a higher number of intermediate activations of the ANN. For MNIST CNN, the AVF value for critical errors is highest for the last FC layer (shown as FC2 above).

Based on these observations, they propose selectively applying TMR to the hidden layer of Flower ANN and the last FC layer of MNIST CNN. They show that their selective TMR approach provides a much better balance between reliability and resource usage than full TMR approach. For example, compared to baseline Iris Flower ANN, selective TMR masks 68% faults with a resource overhead of 45%, whereas full TMR masks 94% faults with an overhead of 200%. For MNIST CNN, selective TMR masks 40% faults with just 8% overhead. In fact, due to the excessive resource requirements of full TMR, MNIST CNN cannot be implemented on the Zynq Ultrascale+ device.

With ongoing process technology scaling, the transistor dimensions and the mutual distances of transistors are reducing [73]. Due to this, the energy from a charged particle may affect multiple nodes, leading to "multi-node upsets" (MNUs). If an MNU happens in multiple nodes of the same cell, it leads to SE in a single bit, but if it happens in multiple cells, it leads to SE in multiple bits. Azizimazreah et al. [39] note that existing fully-hardened SRAM cells are resistant to SEs produced by "single-node-upsets" only (SNUs) and have limited resistance to SEs due to "multiple-node-upsets" (MNUs). Further, these hardened cells have high leakage power consumption. They propose a low-power "fully-hardened SRAM cell" termed "ZMT" (zero-bias MNU-tolerant) which is useful for designing on-chip memory of DNN accelerators.

They note that a large fraction of cells in both feature maps and weights store zero values. Further, $0 \rightarrow 1$ bit-flips have higher impact on DNN accuracy than $1 \rightarrow 0$ bit-flips [4]. Based on these observations, their proposed ZMT cell is immune to SNU for both zeros and ones and MNU for zeros. As shown in Fig. 9, the ZMT cell has two cores, and this redundancy helps in recovering the value of faulty nodes through the help of faulty-free nodes. Each core has two complementary storage nodes. The stored bit is written to the cell using access transistors (J20 and J21).

For avoiding the propagation of node upsets from a faulty node to its complement in the same core, nodes in a core drive the "driver transistors" of nodes in the "opposite core". For instance, nodes SR and SRB drive the "driver transistors" (J10 and J11) of node A.

When a particle strikes the source/drain of an "OFF" transistor, it produces the MNU or SNU at one or more node(s). A strike on an "OFF" PMOS transistor creates a $0 \rightarrow 1$ SNU/MNU and that on an "OFF" NMOS transistor creates a $1 \rightarrow 0$ SNU/MNU. ZMT uses NMOS transistors for driving the storage nodes, e.g., SR and SRB drive the nodes in the upper core, and P and Q drive the nodes in the lower core. This avoids the possibility of any $0 \rightarrow 1$ SNU/MNU, e.g., when a cell stores zero, the only MNU possible is due to the simultaneous $1 \rightarrow 0$ upsets at both Q and SRB.

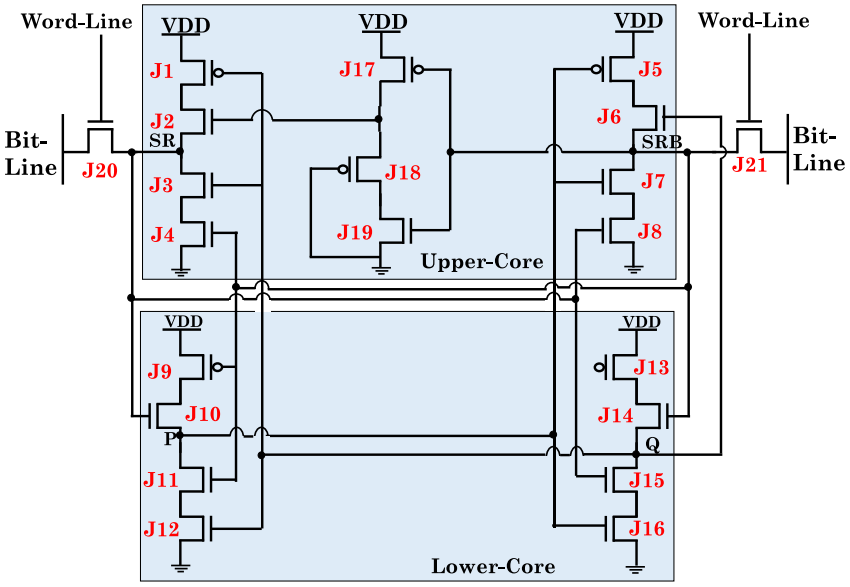


Fig. 9. The ZMT cell design proposed by Azizimazreah et al. [39].

Table 10
Four possible particle strike events in a node of ZMT cell [39] and the result.

Bit stored in cell	Event	Result
0	1 → 0 SNU happens at the node SRB or the node Q.	SNU eliminated
1	1 → 0 SNU happens at the node SR or the node P	SNU eliminated
0	MNU happens at the node SRB and the node Q	MNU eliminated
1	MNU happens at the node SR and P.	Cell flips to 0 (error occurs)

Further, use of NMOS transistors ensures that a 1 → 0 upset cannot turn ON any NMOS "driver transistors" at the nodes for changing the voltage of the nodes, e.g., when a cell stores zero value, a 1 → 0 upset at Q cannot affect the voltages of nodes SR and SRB. The ZMT cell design also ensures that a 1 → 0 upset is not possible at the output. Thus, ZMT cell is immune to MNU when it stores a zero value. However, when an MNU occurs on the cell storing a value one, the cell value is flipped to zero. Table 10 summarizes the outcome of four possible cases of particle strikes in a node of ZMT cell [39].

ZMT cell greatly decreases the "leakage current" when the zero value is stored in it. Compared to DNN accelerators using traditional and full-hardened memory cells, their technique saves a large amount of leakage power. Further, their technique removes 99.99% of incorrect results arising due to soft errors.

Mahdiani et al. [55] note that error-tolerant nature of NN applications allows relaxing the strength of fault-tolerance schemes. For example, for an N-bit adder, they propose protecting only $N - AUC$ MSBs and not protecting the remaining AUC (adder unprotected count) bits. This is shown in Fig. 10(a). By changing the value of AUC, a trade-off between the strength of fault-tolerance and its overhead can be obtained. For protection, they use TMR scheme.

Similarly, a partially fault-tolerant multiplier can be designed, and an example of a 5x5 multiplier is shown in Fig. 10(b). A multiplier with a Q-bit multiplicand and P-bit multiplier has (1) QxP cells termed "carry-save adder" (CSA) array (shown as circles) and (2) a Q-bit vector merging adder (shown as rectangles) for converting the "carry-save redundant" format into "regular binary" format. Every cell has a 2-input AND gate that computes the "partial product" of its input bits and a full-adder that adds the partial products to the running sum. The multiplier has two

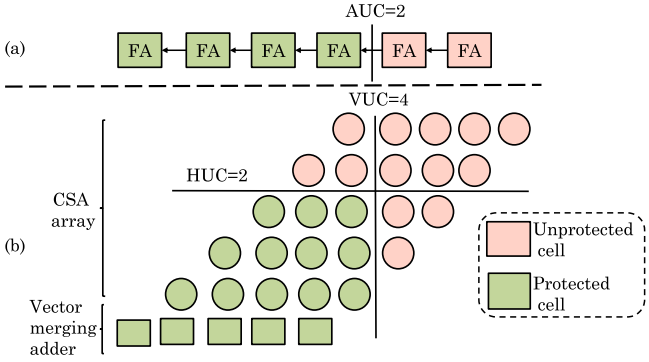


Fig. 10. (a) A relaxed 6-bit adder with AUC=2 [55] (FA = full adder, CSA = carry-save adder) (b) A relaxed 5x5 multiplier with HUC=2, VUC=4 .

parameters for controlling which cells are and are not protected. These are termed "horizontal unprotected count" (HUC) and "vertical unprotected count (VUC), and they are shown in Fig. 10(b). Increasing the value of HUC/VUC moves the boundary line down/left, respectively.

They employ their proposed adder/multiplier for a hardware accelerator of a face-recognition ANN. They find that the minimum word length for a bit-true implementation to achieve the same performance as the FP implementation is 13 bits. They compare their technique with an implementation which provides full protection using TMR. Faults are injected only during the testing phase. They compare both the final recognition accuracy and "average mean squared error" of all the network outputs for all the training and testing patterns. Use of these metrics allows evaluating how well their technique approaches the external and internal behavior of full TMR.

They find that with rising AUC, there is a linear increase/decrease in the number of soft-errors in unprotected/protected area, respectively. The trends observed on changing HUC are similar, although the magnitude of change in SEs is much larger. This is because changing AUC by one removes/adds just one full-adder from the protected region. By comparison, changing HUC changes the protection strength of all the CSA cells of a partial product, and hence, changing it has a greater impact on the SE rate. The same also applies to VUC. As the probability of occurrence of a fault on a signal (P_e) increases from 10^{-8} to 10^{-5} to 10^{-4}, the value of AUC required for their technique to achieve same "accuracy" as full TMR is 13, 7 and 3 bits, respectively. To achieve same

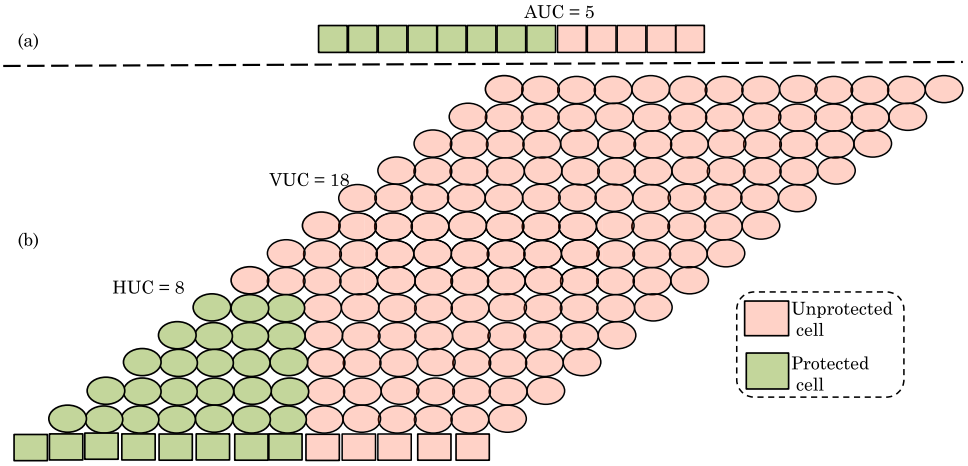


Fig. 11. (a) Relaxed adder with AUC=5 [55] (b) relaxed multiplier with HUC=8, VUC=18.

Table 11
Largest value of AUC/HUC/VUC for the technique of Mahdiani et al. [55] to follow full TMR scheme.

	$P_e = 10^{-8}$	$P_e = 10^{-5}$	$P_e = 10^{-4}$
AUC	13	7	0
HUC	13	10	0
VUC	26	20	0

"internal behavior" as full TMR, AUC needs to be 13, 7 and 0 bits, respectively. The lower of these two is taken as the final AUC. Similarly, the values of HUC and VUC are found, and they are shown in Table 11.

Since a fault in multiplier affects only one multiplication, whereas a fault in adder affects the accumulation result of all the earlier multiplications, a higher degree of protection is required for the adder than for the multiplier. Hence, although increasing AUC leads to fewer faults than that caused due to increasing HUC/VUC, the faults in adder have a higher impact on the output quality. Hence, they propose choosing a smaller value of AUC and higher values of HUC/VUC.

When AUC, HUC, and VUC are varied simultaneously, for their technique to achieve the same behavior as full TMR, the values of (AUC, HUC, VUC) should be (5,8,18) at $P_e = 10^{-5}$ and (0,0,0) at $P_e = 10^{-4}$. The diagram of adder/multiplier for (5,8,18) is shown in Fig. 11. At $P_e = 10^{-5}$, their technique reduces the area by 250% and critical path delay by 50% compared to full TMR. However, at high error rate such as $P_e = 10^{-4}$, their technique becomes similar to the full TMR and thus, offers no advantage over full TMR.

4.4. Techniques for DRAM memory of DNN accelerators

Nguyen et al. [41] present a technique to save DRAM energy by reducing the refresh rate. Their technique stores the data in memory in a transposed manner. Fig. 12 illustrates their technique with a simplified example of storing byte-granularity data. In the baseline memory shown in Fig. 12(a), address 0 stores Value[0], address 1 stores Value[1], and so on. In their design shown in Fig. 12(b), address 7 stores the MSB of all values, address 6 stores the second-MSB of all values, and so on. Then, the refresh rate of DRAM cells at addresses storing few LSBs is reduced to save energy. The faults introduced due to the reduced refresh rate can be tolerated due to the resilience properties of DNN.

In the actual DRAM architecture, assuming only 32 rows per bank, an FP32 no. is stored such that row 31 stores the MSB and row 0 stores the LSB, and so on. Fig. 12(c) illustrates this design. The first nine rows are refreshed at a normal rate since they store sign and exponent bits which are crucial. The remaining 23 rows are refreshed at increasingly larger periods since they store mantissa bits of decreasing importance. The refresh period of these rows is increased linearly.

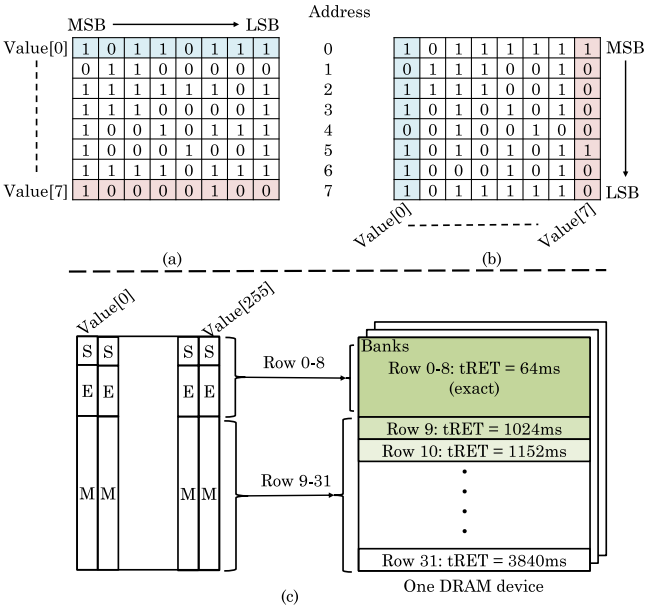


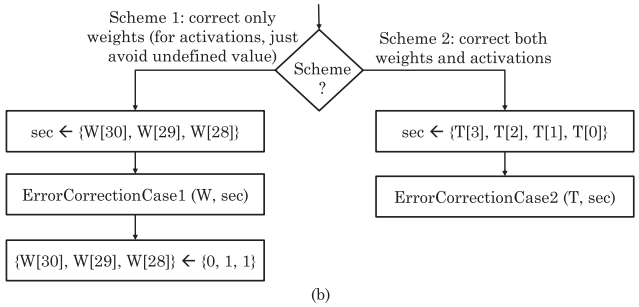
Fig. 12. (a) Traditional data storage approach (b) significance-based data storage approach proposed by Nguyen et al. [41] (c) row-level refresh technique (S/E/M=sign/exponent/mantissa) .

A limitation of the proposed design is that when a single value is referenced, 32 addresses need to be accessed. To mitigate this issue, they propose "blocked accessing scheme" whereby 32 data-items are accessed simultaneously and are transposed before sending them to the memory controller. The data to be transposed is temporarily stored in a buffer. For 32b data and 8b DRAM burst length, single data access leads to 256b transfer. Hence, the buffer size needs to be 256x32bits. Data-transfer between the buffer and DRAM requires 32 commands which incur delay penalty. This overhead can be alleviated by using a prefetching scheme based on the predictable access pattern of DL applications. Their technique saves a large fraction of DRAM energy while incurring a negligible loss in DNN classification accuracy.

Nguyen et al. [18] exploit properties of DNNs for improving DRAM resilience. In their design, the DRAM has two regions: (1) a fault-free region refreshed at a normal rate which stores critical data, such as control data (2) a faulty region refreshed at a lower rate depending on the BER level that can be tolerated. This region stores non-critical data such as DNN parameters. They primarily focus on the training phase and hence, assume that the data structures are stored in FP32 (single-precision) format.

	Scheme-1	Scheme-2
Correction data stored in bits	{30 29 28}	{3 2 1 0}
Bits which receive protection	{31 27 26 25}	{31 30 29 28 27 26 25 24 23}

(a)



(b)

Fig. 13. (a) The bits which are protected or corrected in different schemes [18] (b) Working flow of two schemes.

They study the distribution of weights and activations in three DNNs: GoogleNet, AlexNet and ResNet50. They find that the absolute values of 99.5% of the weights are in the range $[2^{-13}, 2^{-2}]$ which implies that the positions {30,29,28} of FP32 are {0,1,1} (refer Table 1). The activations of GoogleNet and AlexNet are in the range $[2^{1}, 2^{11}]$ which implies that bits {30,29,28} of FP32 are {1,0,0} (refer Table 1). However, the 99.9% activations of ResNet50 fall in the range $[2^{-12}, 2^{11}]$. The above distribution occurs due to the regularization used for limiting the range of weights to [-1,1] during training. By comparison, activations are obtained from multiplication and hence, they are generally larger than 1.

Based on these observations, they propose two schemes, which are summarized in Fig. 13:

1) Scheme 1: Here Hamming(7,4) code bits are stored in three bits {30,29,28} and they are used for correcting {31, 27, 26, 25} bits of the FP32 weight. After error correction, value of {30,29,28} bits are assigned to {0,1,1}. Thus, the exponent is set to 011xxxxx implying that all weights are within $[2^{-13} : 2^{-2}]$. In this scheme, errors can happen in only two bits of the exponent (24th and 23rd). This error can increase/decrease the weights by at most 2^{2}. This is much less than a factor of 2^{16} change due to an error in 27th bit on using no protection. A factor of 2^{2} error can be alleviated during the training. Scheme-1 does not protect activations since they are much more resilient than the weights [16]. Only if their value becomes undefined, their 30th bit is changed to 0 to remedy this error. During inference, only weights are stored in memory. Activations are repeatedly replaced by new data and stay in memory for a very short period. Hence, activations need not be protected.
2) Scheme-2: it stores Hamming(13,9) bits in four LSBs of FP32, and they are used to protect bits 31st to 23rd of FP32. This scheme can be used to protect both weights and activations.

They evaluate the efficacy of their technique for both the training phase and the inference phase. (1) Results on training phase: during the training phase, fault injection on weights is done during both FWP and BWP and on activations during BWP. While JEDEC ("joint electron device engineering council") standard prescribes BER to be 10^{-15} for safe operation, their technique can work with BER as high as 10^{-7} and still provide same training accuracy as the baseline technique which performs refresh operations at normal rate. Also, by reducing the rate of refresh operations, their technique saves energy and improves training speed. (2) Results on inference phase: during the inference phase, fault injection is done on weights only. They find that scheme-1 provides higher accuracy than scheme-2.

Kim et al. [25] present a technique for improving the reliability of die-stacked memory, viz., "high bandwidth memory" (HBM) based on the properties of DNNs. Unlike for planar memories, in die-stacked memories, different dies have asymmetric SE rate. Due to alpha particles arising from upper packaging material, the top-most die shows higher SE rate than the remaining dies. Also, due to the heat released from the interposer and logic die, the bottom-most die shows a high rate of SE.

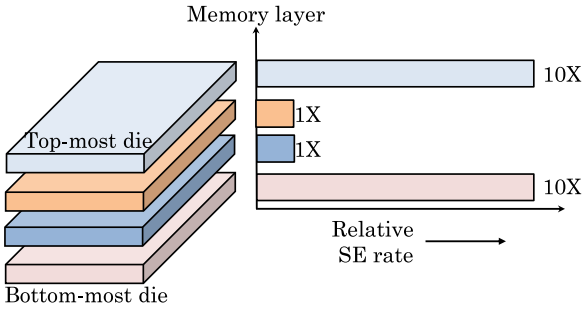


Fig. 14. Relative SE rate of different dies in a die-stacked DRAM [25]. The top-most and bottom-most layers are termed as vulnerable dies and the middle layers are termed as normal dies.

Due to these factors, the SE rate of the upper-most and lower-most dies are 10 × higher than that of remaining dies. This is shown in Fig. 14.

They further find that all the weights of AlexNet are below 1. When expressed in FP32 format, the 30th and 29th bit of all the weights are 0 and 1, respectively (refer Table 1). Hence, they study the impact of the following two cases

- Case-A: flipping 29th bit (1 → 0) of 500 weights in each layer of DNN
- Case-B: flipping 30th bit (0 → 1) of 5 weigths in each layer of DNN.

They find that the accuracy loss of AlexNet is higher for Case-B than for Case-A, even though less number of faults are injected in case-B. This could be because of one or both of the following reasons: (1) a flip in higher-order bit has a higher impact (2) a 0 → 1 flip has a higher impact than a 1 → 0 flip. Of the two, they find that the latter reason has a higher impact on accuracy, as confirmed below.

Apart from the 30th bit, they find the second 0 bit in the higher bits of exponent, which is termed as "second-zero location bit" (SZLB). For example, from Table 1, the exponent bits of the number 0.125 ($= 2^{-2}$) are 01111100 and hence, its SZLB is the 24th bit (refer Fig. 2(a)). As another exmaple, for the number 0.00390625 ($= 2^{-8}$), the exponent bits are 01110111, and thus, its SZLB is the 26th bit. SZLB is important since flipping of it also has a high impact on accuracy. They find that the SZLB of most weights of different DNNs is as follows: 26th bit for VGG16 and AlexNet, 25th bit for pruned VGG16 and pruned AlexNet. For remaining weights, SZLB is generally the 27th bit. Thus, for all four DNNs, SZLB is generally found at 27th, 26th or 25th bit and hence, a fault in these bits can have a high impact on accuracy. They define 30th bit as the "most unreliable bit" (MUB) and 27th, 26th and 25th bit as the "second-most unreliable bit" (SMUB).

Based on these observations, they propose masking the MUB which avoids 0 → 1 flips in MUB completely. Further, an FP32 value except MUB is partitioned in two parts: SMUB portion and non-SMUB portion. SMUB is stored in the normal dies, and non-SMUB is stored in vulnerable dies. Table 12 summarizes their observations and the proposed protection techniques. This technique is implemented on an HBM logic die.

Table 12
Observations and proposed techniques of Kim et al. [25].

Observations	Terminology	Protection technique
Weights of AlexNet are below 1 ⇒ W[30]=0 and W[29]=1	30th bit is termed MUB	Mask MUB
SZLB is 26th bit for most weights of VGG16 and AlexNet and SZLB is 25th bit for most weights of pruned VGG16 and pruned AlexNet. For the remaining weights, SZLB is 27th bit.	27th, 26th and 25th bits are termed as SMUB	Store SMUB in normal dies
Remaining bits are not critical	Remaining bits are non-SMUB	Store non-SMUB bits in vulnerable dies

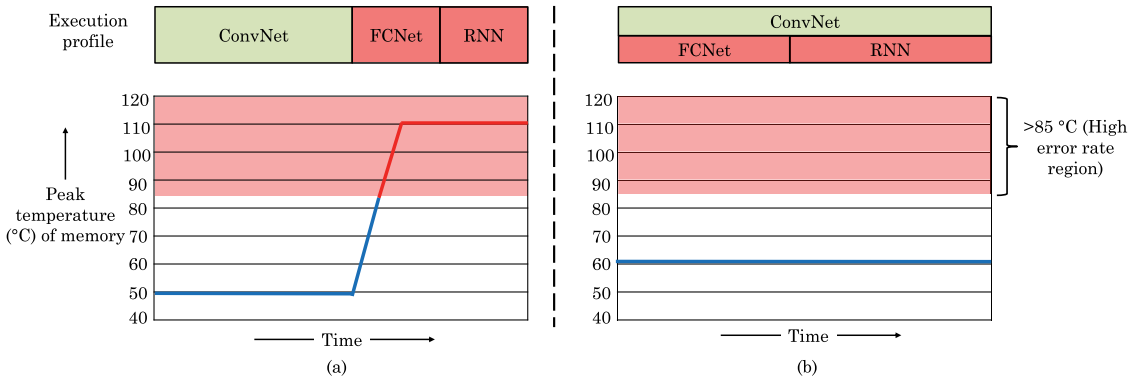


Fig. 15. (a) Executing ConvNet and FCNet/RNN sequentially leads to high memory temperature when FCNet/RNN are executed since they are memory-intensive (b) The technique of Yin et al. [35] executes ConvNet and FCNet/RNN in parallel to achieve uniform temperature profile.

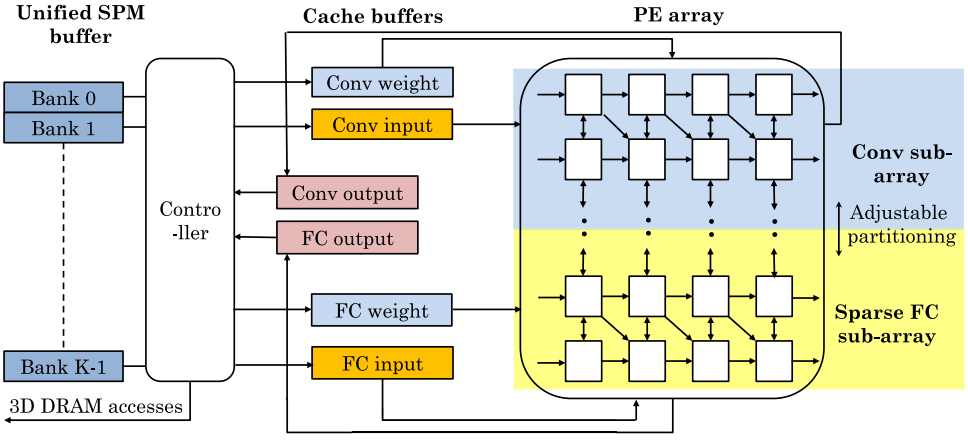


Fig. 16. The architecture proposed by Yin et al. [35] .

HBM has two channels in each die. They store partitioned data in every memory die and assign two channels for each weight. For 1% accuracy loss limit, their technique can tolerate as much as $10^4\times$ higher BER than a traditional technique.

Reliability issues on running hybrid NNs on accelerators with stacked memory: In real-world applications, multiple NNs viz., ConvNet, RNN, and FCNet are cascaded, which leads to a hybrid-NN architecture. For example, AlexNet may be used for image classification, and "long-term recurrent convolutional networks" (LRCN) may be used for image understanding. Hybrid-NNs have unique characteristics. The core computation pattern of ConvNet is convolution, whereas that of RNN and FCNet is FC computations. The memory requirement of ConvNet is small, whereas that of RNN and FCNet is high. Thus, executing these NNs in a serial-order leads to non-uniform memory accesses. This creates high peak temperature in a "3D + 2.5D stacked" accelerator design where a 3D stacked DRAM is used alongside a DNN accelerator on a silicon interposer [74]. This is illustrated in Fig. 15(a). High temperature increases the cooling overhead, number of DRAM faults and requires increasing the refresh rate [21].

Yin et al. [35] propose a technique for addressing thermal issues while executing hybrid-NN on a neural processor with "3D + 2.5D" stacking. Their proposed architecture is shown in Fig. 16. It uses a PE array which can be flexibly partitioned into two portions: one portion accelerates CONVNet and second portion accelerates sparse FC computations (FCNet and RNN). For CONV mode, systolic dataflow is used. Further, their technique uses a "unified SPM buffer" designed with several SRAM banks for high-parallelism. The inputs and weights used by PE array and the output produced by the PE array are stored in the buffer. The size of inputs/weights/outputs can vary for different layers, and this allows full utilization of the buffer. The total memory accesses are reduced due to improvement in data reuse through tiling and scheduling scheme for on-chip SPM buffer. For bridging the dataflow between the PE array and SPM buffer, "cache buffers" are used. Cache buffers store the data of SPM buffer and help in concurrently serving various types of requests viz., weights/inputs/outputs. Unlike SPM buffer, cache buffers do not take any space in the memory address space.

Their technique avoids bursty memory accesses using "spatial division mapping" (SDM). The key idea of their technique is shown in Fig. 15(b). The bandwidth requirement of CONV is different from that of sparse FC. Hence, their SDM technique, which executes ConvNet and sparse FC in parallel, reduces the peak bandwidth demand. There are three reuse patterns: input, weight, and partial sum reuse, and each of

them is good for different types of layers. For example, input reuse is good for the layers where the footprint of input data exceeds that of weight and partial sum. Hence, their technique uses different scheduling schemes for different layers to optimize data reuse. For ConvNet, their technique uses a batch size of one, whereas, for RNN and FCNet, larger batch size is used.

Their technique parallelly executes ConvNet and RNN/FCNet, and thus, the inference latency is the larger of the two latencies. Their compiler finds a suitable partitioning of the PE array and SPM buffer between ConvNet and RNN/FCNet to optimize two goals (1) minimizing overall inference latency and (2) minimizing peak bandwidth demand. In summary, their technique reduces *total memory accesses* by exploiting data reuse and *peak memory accesses* by leveraging the complementary memory access behavior of various NNs. Their technique reduces both steady-state and peak temperature with no performance loss.

4.5. Techniques for voltage-scaled DNN accelerators

"Voltage underscaling based timing speculation" works on the observation that in digital logic, worst-case "timing critical paths" are used infrequently. Hence, if the TEs can be handled, the system can be run at supply voltages below the normal voltage for saving energy. There are two approaches for handling timing errors: (1) "timing error detection, and recovery" (TED) scheme identifies the TEs using Razor flip-flops and corrects them by re-executing the corresponding input. (2) "timing error propagation" (TEP) scheme lets the errors move forward and relies on the error-tolerance of the algorithm.

The limitation of TED scheme is that with the rising number of MACs in a TPU-like DNN engine, the probability of at least one MAC showing an error increases fast. In a systolic array, an error in even one MAC can force all the inputs to stall due to the requirement of maintaining synchronism. Thus, TEs translate into severe latency and energy penalties. In fact, due to the deeply pipelined design of TPU, TEs also lead to huge loss in classification accuracy of the DNN executing on the TPU [48]. Similarly, TEP scheme can lead to a large loss in the accuracy of DNN because the errors usually affect the MSBs of MAC outputs. Thus, both the TED and TEP schemes have limitations.

Zhang et al. [14] present a technique, termed TE-drop ("timing error drop"), for allowing aggressive voltage underscaling in DNN accelerators without degrading accuracy. They assume a TPU-like DNN accelerator. TE-drop uses Razor flip-flops for detecting TEs, but unlike TED, does not re-execute the faulty MAC operation for recovering from TEs. Since the weights of a DNN are biased towards small values, each MAC operation makes an only a small contribution to the output of a neuron. On a TE in a MAC, TE-drop technique steals the subsequent clock cycle from its downstream MAC for correctly finishing its own update to the "partial sum". It also skips the update of downstream MAC. This is illustrated in Fig. 17(a).

Apart from "Razor flip-flops", TE-drop technique also uses a MUX, whose output is decided by the "error signal" from the previous MAC unit. If the previous MAC unit had a TE, the MUX passes on previous MAC's correct value of partial sum which is received from Razor's shadow flip-flop. However, if the previous MAC had no TE, the present MAC unit computes the "partial sum", as usual. This design is illustrated in Fig. 17(b).

In effect, TE-drop technique randomly drops some connections between two neurons where the randomness comes due to TEs. For avoiding overfitting, techniques such as dropout and dropconnect are anyway used in DNN training which randomly drop neurons and connections. The TE-drop technique is similar to these techniques and allows operating at more than 10% TE rates without loss in accuracy.

They further find that different layers show different TE rates on using the same voltage underscaling ratio. For larger networks such as AlexNet, the TE rate reduces with depth, whereas for smaller networks such as Reuters, the TE rate is highest for deeper layers. They propose an algorithm that applies a different degree of voltage underscaling

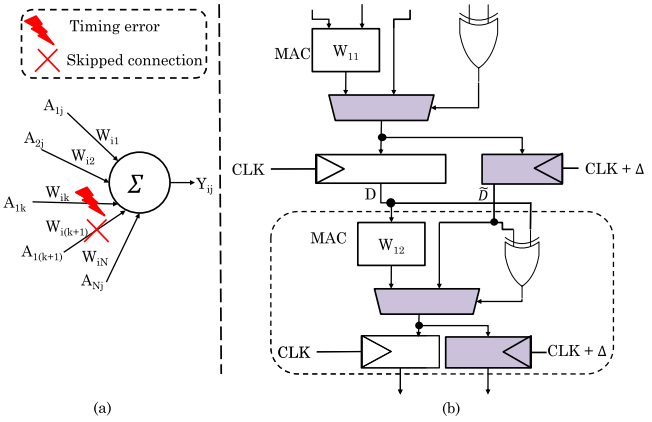


Fig. 17. (a) In TE-drop technique [14], on a TE in a MAC, update of its subsequent MAC is skipped (b) circuit-design of TE-drop technique.

to different layers such that the TE rate contribution of all the layers is nearly the same. This technique shows lower TE rate than a simple technique that applies uniform voltage underscaling to all the layers.

Compared to a technique which avoids performing MAC operation if either activation or weight is zero, TE-drop technique provides large energy-saving without any loss in performance and less than 1% loss in accuracy. The TE-drop technique reduces the energy of non-zero MAC operations by voltage underscaling, and thus, TE-drop technique can be used along with the technique of avoiding MAC operations leading to zero output.

The limitation of TE-drop technique [14] is that using it for mitigating a TE in the last row of MACs leads to large performance loss at NTV. Since the values of partial sums are highest at the bottom of the systolic array, the impact of TEs is also highest in the last row of MACs. Further, with increasing fault rate, skipping the update of some MACs reduces the final accuracy significantly.

Choi et al. [42] present a technique for improving the reliability of DNNs based on the sensitivity of different neurons. The sensitivity of a neuron is defined as the absolute change in its cost on introducing a noise. By using the first-order Taylor expansion [75], the sensitivity can be approximated to be the product of the gradient and the noise. They express noise as the function of BER value, and the gradient is computed by differentiating and then averaging over the target variable for the whole dataset. They show that the sensitivity obtained from the fault-injection method is close to that obtained from their approach, especially for small values of BER. They also find that weights with small magnitude have high sensitivity, whereas those with larger magnitude do not have high sensitivity.

They assume a TPU-like systolic array. The architecture with their proposed changes is shown in Fig. 18. They insert Razor flip-flop between the multiplier and the adder, which works as an extra pipeline stage. If a TE is detected in the multiplication, the multiplication output is ignored, and the unmodified partial sum is passed to the next MAC unit. Although this change increases the latency by a single clock cycle, the throughput of the systolic array is not affected.

An ideal mapping would assign weights with higher sensitivity to more robust MAC units and vice versa. Since this may require large interconnect overhead and non-trivial changes to the hardware, they also propose other mapping strategies. For example, consider the naive "column-wise mapping" scheme shown in Fig. 19, where each filter is mapped to a MAC column. Their proposed mapping technique assigns filters with high sensitivity weights to the MAC column having lower mean values of TE rates. Thus, unlike the ideal mapping which decides the optimal mapping of "individual weights" to "individual MACs", this technique performs mapping of each "filter" to one "column of MAC". By virtue of accounting for sensitivity of different neurons, their technique

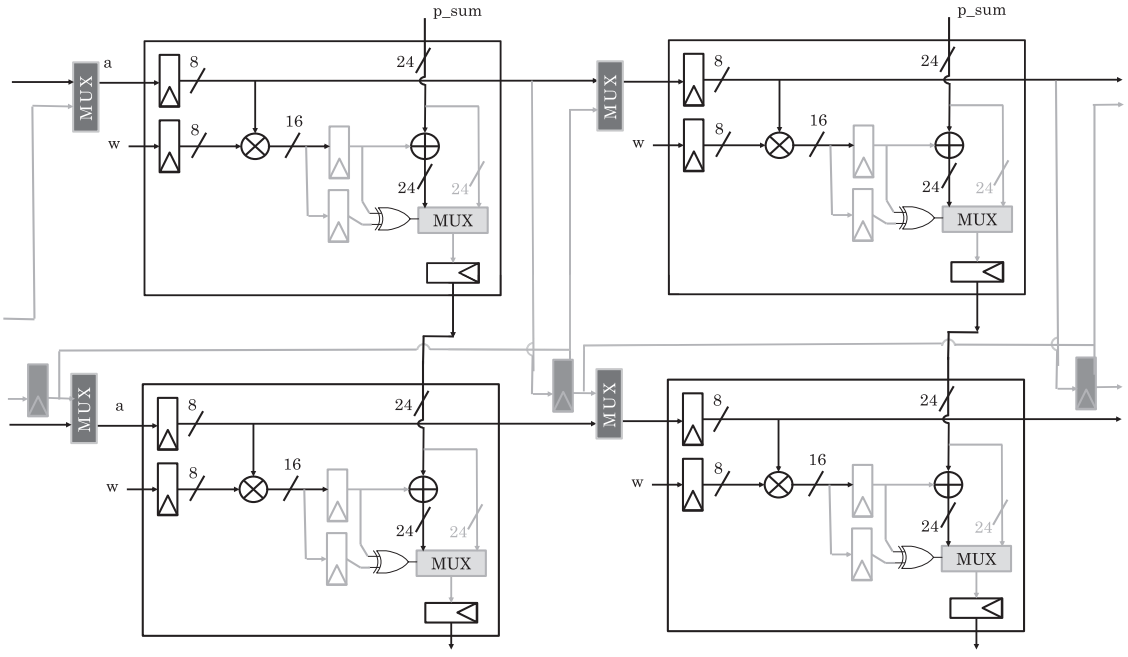


Fig. 18. The architecture proposed by Choi et al. [42] (p_sum=partial sum).

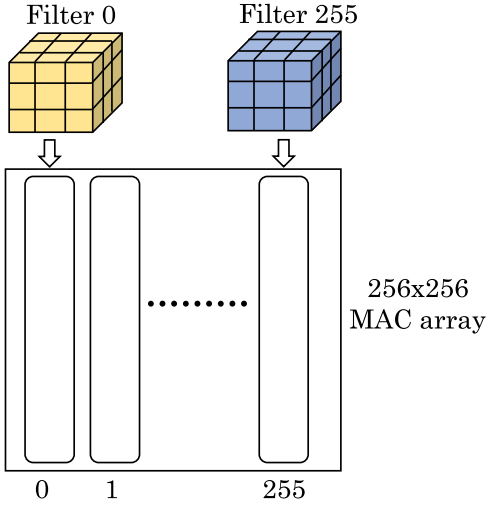


Fig. 19. A naive strategy where DNN filters are mapped to MAC columns based on their ID. The mapping proposed by Choi et al. [42] ranks the filters based on the sensitivity of their weights, ranks the MAC columns based on the mean value of their TE rates, and then maps most sensitive filters to MAC columns showing least TE rates.

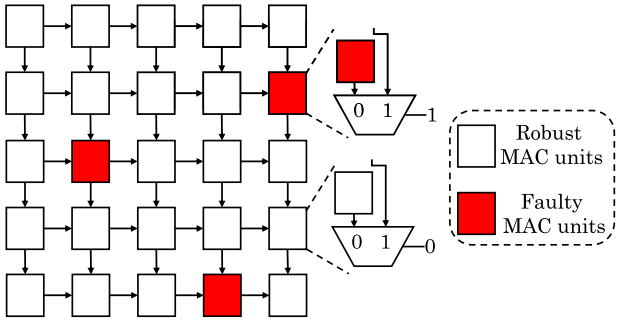


Fig. 20. Fault-aware pruning scheme proposed by Zhang et al. [12] .

provides higher energy saving than the TE-drop technique [14] for the same accuracy loss.

Zhang et al. [12] study the reliability of a TPU-like accelerator. They study permanent faults in data-path and not in memory or control logic. They note that there is a large degradation in the accuracy of TPU even when four out of 64K MACs of TPU are faulty. This is because the faults change the higher bits of MAC output and hence, the faulty outputs are typically much larger than the correct outputs. They propose two techniques for recovering the accuracy loss due to faults.

They note that in TPU, the mapping between DNN weights to MAC units is fixed. Based on this, their first technique prunes (i.e., sets to zero) any weight mapped to a faulty MAC unit. If multiple weights are mapped to a faulty MAC unit, then all those weights are pruned. Hardware implementation of pruning is done by adding a bypass route around the faulty MAC unit, as shown in Fig. 20. On enabling of this route, the value of the faulty MAC is not accounted in the column sum. The area penalty of bypass route is below 10%. Note that their fault-aware pruning scheme is similar to the word-masking scheme [49,51] since both these schemes set a faulty weight to zero. The difference is that Zhang et al. perform pruning on permanent faults in the MAC unit, whereas Reagen et al. [49] perform word-masking on transient faults in SRAM memory arising due to voltage scaling.

Their second technique works by applying the first technique and then performing retraining of unpruned weights to recover the loss in accuracy. During retraining, the pruned weights remain zero. The retraining is done only once for every TPU chip. Their techniques allow operating the TPU when as high as half the MAC units are faulty, and the loss in accuracy remains insignificant.

Reagen et al. [49] present a technique for optimizing DNN accelerators. Their technique first selects DNN hyperparameters that minimize prediction error without requiring excessive resources (such as memory consumption). The weights chosen are kept fixed for further experiments. Then, optimal microarchitectural parameters such as memory bandwidth, loop-level parallelism, and clock frequency are selected using cycle-accurate simulation. The design which balances energy and area is used as the baseline design.

They further propose three optimizations: (1) Quantization: They use FxP representation shown as (m, n), where m and n show the number

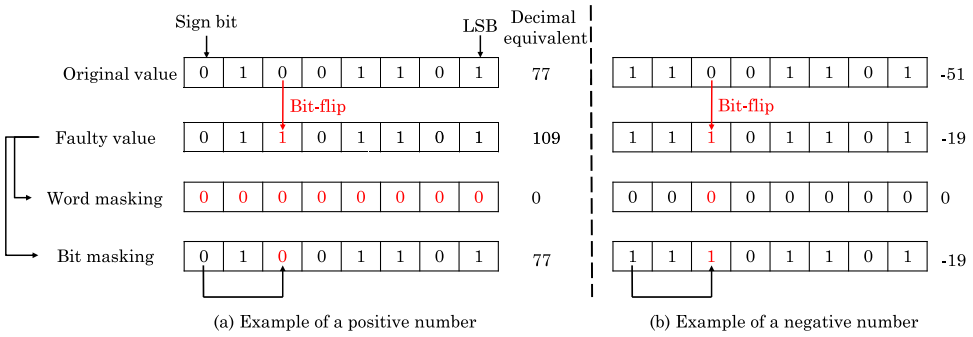


Fig. 21. Illustration of word-masking and bit-masking schemes [49] for (a) a positive number and (b) a negative number (in2's complement format) .

of integer bits (including the sign bit) and fractional bits, respectively. For every layer, they find the minimum value of (m, n) individually for weights, activations, and partial products. The minimum value is found such that reducing the value of m or n by 1 bit exceeds the error confidence interval. Since different DNN layers are processed on a DNN accelerator in a time-multiplexed manner, they set the width of all datatype to the highest requirement for any layer. For example, if the weights of layer 0, 1, 2, 3 require 8, 7, 7, 6 bits for ($m + n$) respectively, then for all the layers, the bitwidth of weights is set to be 8 bits. This saves area by avoiding the need of having different word sizes for every layer. They find that compared to using (6,10) bits for all datatypes, using (2,6) for weights, (2,4) for activations and (2,7) bits for partial products provides considerable power savings.

(2) Ineffectual operation skipping: they find that 75% activations have a magnitude below 1.05 and avoiding the corresponding SRAM read and MAC operation has no impact on the output. The scope of operation skipping is high because the ReLU function removes negative products. Further, DNNs using ReLU become increasingly sparse with deeper designs due to successive decimation. Since the value of activations depends on the input, the operation to be skipped cannot be determined a priori. Hence, their technique performs fetch operations in two stages. First, the activation is read from SRAM and compared against a per-layer threshold. If the activation is below a threshold, a flag bit is set. The weight read is predicated on this flag and the subsequent MAC operation is stalled using clock-gating to save power.

(3) SRAM voltage scaling: After applying above optimizations, SRAM power accounts for most of the remaining power. Hence, they perform SRAM voltage scaling. Compared to using 0.9V supply, using 0.7V supply reduces the power consumption by half and leads to a bitcell fault rate close to 10^{-5}. To detect these faults, they use Razor double-sampling scheme [76]. This scheme monitors every column of the array individually and hence, it can detect any number of faults, and it can also identify the location of the faulty bit. It incurs nearly 13% power and negligible area penalties. Since correcting the faults using conventional techniques such as replay incurs large overhead, they present a simple scheme for mitigating the impact of faults on accuracy. Specifically, on detecting a fault, they mask the data towards zero. Masking is done at either of the two granularities: word-masking where all the bits of a faulty word are put to zero and bit-masking where any faulty bit is replaced with the sign bit. These schemes are illustrated in Fig. 21 with the example of a positive number and a negative number.

Word masking makes the faulty weight as zero and thus, removes the corresponding connection from the DNN. This prevents the fault from propagating. However, zeroing a large weight value can have a high impact on the activation values of the subsequent layer. The advantage of word masking is that it causes a constant error regardless of the number of faults. This is because any number of non-zero faults leads to resetting of the register to zero value. Word masking enables tolerating an order of magnitude higher bitcell faults than the scheme using no protection.

Bit masking reduces the magnitude of a faulty weight by virtue of rounding a faulty bit position towards zero. This is corroborated by the examples shown in Figs. 21(a) and 21(b), where bit-masking reduces of maintains the magnitude of a faulty value. Since it masks only the affected bit, it provides stronger protection than word masking. It limits fault propagation through FC layers, and its bias towards zero is complementary to the sparsity of DNNs using ReLU activation function. Compared to word masking, 44 × higher faults can be tolerated in weight SRAMs on using Razor scheme and bit-maksing. This allows further reducing the supply voltage by 0.2V without loss in accuracy, which further reduces the power consumption. Overall, their technique brings DNN power consumption to tens of milliwatts. A limitation of the bit-masking technique is that it offers no protection to the sign-bit. Further, since other bits are masked with the sign-bit, a fault in the sign-bit can propagate to other bits, leading to a large amount of error.

Salami et al. [51] propose a fault-mitigation technique which applies three schemes, in that order:

1) On a fault in the MSB, it is masked with the sign-bit. This is based on the observation that for most of the times, the values of sign-bit and MSB are the same. This is because most of the NN data is close to zero and in two's complement format, for *small-magnitude* positive numbers, SignBit = MSB = 0 and for *small-magnitude* negative numbers, SignBit = MSB = 1.
2) Using the sign-bit as a mask, non-sign bits are recovered using the Bit-masking scheme [49].
3) If the sign-bit is also faulty, the register is reset to zero using the word-masking scheme [49].

Compared to using either bit-masking and word-masking technique [49], their technique lowers the inference error.

Chandramoorthy et al. [19] present a technique for enabling the low-voltage operation of SRAM-based accelerator for DNNs. They first study the impact of faults in different data types on the accuracy of a fully-connected DNN. They note that injecting faults in the weights of all the layers leads to a much larger loss in accuracy than injecting faults in the input image. Further, faults in the weights of the first layer have a higher impact than those in the last layer. This is because the first layer has a higher number of weights and faults in them accumulate until it reaches the output.

To ensure that the loss in accuracy compared to that at a nominal voltage (0.5V) is at most 2%, the supply voltage needs to be 0.46V or above. Their proposed booster circuit achieves the target voltage of 0.46V from very low supply voltages (e.g., 0.34V). Their technique controls the voltage during every read/write operation with no latency penalty. The other idle SRAMs remain at unboosted supply, which saves leakage energy. Their technique boosts the voltage of the SRAM power grid only. The logic always operates at lower supply voltage. At the circuit level, boosting is performed by increasing or reducing the number of "boosters" or the magnitude of "boost capacitance" connected to the voltage supply.

The supply voltage of each SRAM bank can be individually boosted to any of the P levels. Further, the level of boosting can be changed in different execution intervals. Based on the accuracy requirement at the

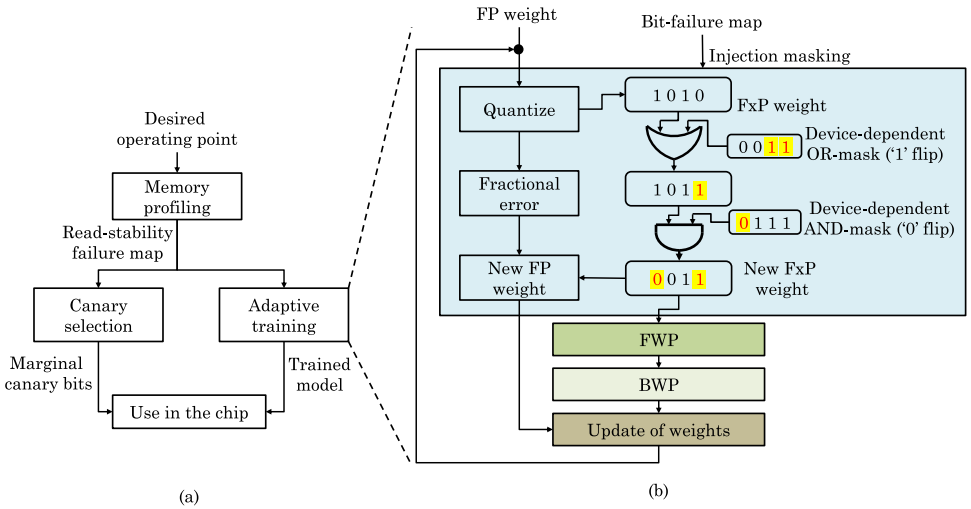


Fig. 22. (a) Overall flow of MATIC [50] (b) Working of injection-masking scheme.

application level, their technique sets disparate boost levels to SRAM cells storing weights of different layers, inputs or interim computations and control bits. For example, the input is boosted to a low voltage just sufficient for ensuring reliable operation. The level of boosting is kept same for accesses in a layer but can be different for those in different layers. Achieving such fine-control is infeasible with DVFS (dynamic voltage/frequency scaling) using voltage regulators since they incur high energy, area, and latency penalty.

Their technique does not require modifying the internals of SRAM. Also, it is independent of the standard cell libraries and the process technology. Their technique can work on 6T, 8T, and 10T SRAM cell designs. At supply voltage above the nominal voltage (0.5V), boosting SRAM voltage reduces its access latency without requiring any redesign of the SRAM array. This is especially helpful for deep pipelined designs. They validate the working of their technique on a chip fabricated in 14nm bulk-FinFET. For this chip, their technique saves 30% energy over the use of single supply for memory and logic, with no loss in accuracy.

Kim et al. [50] present a technique, named MATIC ("memory adaptive training with in-situ canaries") that allows aggressively scaling the voltage of weight SRAMs. In 6T SRAM, with decreasing voltage, the noise margin is reduced, and then, the bitcell gets flipped to its "preferred state" during a read operation. Once flipped, the bit-cell maintains the state in later reads and favors its (wrong) value. Thus, although the spatial distribution of the read-disturbance failure at low voltage is random, the read output of a particular bitcell itself remains the same as its preferred state. Fig. 22(a) shows the overall flow of their technique. Their technique first performs SRAM profiling by performing a "read-after-write" and a "read-after-read" operation on every SRAM address. From this, the preferred state of each bitcell is found, and this creates a failure map.

This information is used during the training process to allow CNN to compensate for the error. Specifically, if the data stored in a bitcell is the complement of its "preferred state", reading at very low voltage would flip the cell. Then, future reads will read wrong values *consistently*. Based on this consistency property, their technique applies bit-masks according to the profiled bit-errors to every weight before FWP. This is shown in Fig. 22(b). Due to the use of bit-error masks, the loss propagated in BWP shows the impact of bit-errors and hence, weight-updates compensate the effect of the error. Due to the SRAM word length limitations, the weights are quantized during training. Their technique preserves the fractional quantization error to alleviate the accuracy loss due to use of quantized weights. This allows performing FP training where weight-updates happen gradually over multiple BWP iterations.

To understand Fig. 22(b), assume that after quantization, an FxP weight value is 1010. Now, assume it is stored in four SRAM bit-cells

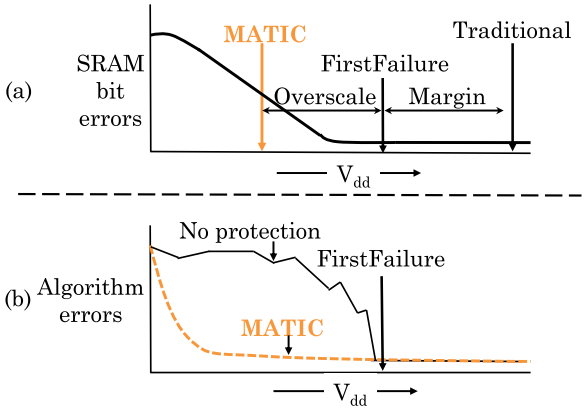


Fig. 23. (a) SRAM error and (b) CNN error at different voltages (Nominal voltage =0.9V, voltage at first failure =0.53V). Without MATIC, nearly all SRAM cells fail at ~ 0.44V. MATIC [50] allows operating at much lower voltage (0.46V) than what naive voltage-scaling approach allows (0.53V).

and their bit locations be termed ABCD. Assume that for a particular SRAM device, locations C and D tend to flip to value 1, location A tends to flip to value 0, and location B does not show any fault behavior. Then, flipping of bits C and D to value 1 is equivalent to doing a logical-OR operation with value 00**11**. Similarly, flipping of location A to 0 can be modeled as doing a logical-AND operation with **0**111. Hence, *for this device*, the OR-mask is 00**11** and the AND-mask is **0**111. On applying these masks consecutively, the value 1010 is finally changed to 0011.

The "in-situ canary circuits" are bit-cells chosen from weight SRAMs, and they allow dynamically controlling the supply voltage. In the conventional canaries, critical circuits are replicated for detecting an imminent failure. However, they need an extra margin and become affected by process-variation. MATIC uses selected weight bit-cells that are on the margin of "read-failure" as in-situ canaries. These cells fail before other crucial cells and maintain their stored values. Since CNNs are tolerant of a small number of errors, their accuracy remains unaffected even when these cells fail, and thus, the supply voltage can be scaled down aggressively. This approach provides reliability margin specific to the failure pattern of the chip which is better than using "static voltage margin". They demonstrate MATIC on a CNN accelerator fabricated in 65 nm CMOS process. Figs. 23(a) and 23(b) provide conceptual illustration of SRAM error and CNN error at different voltages. MATIC allows significant overscaling without losing CNN accuracy and also achieves substantial energy saving compared to normal voltage.

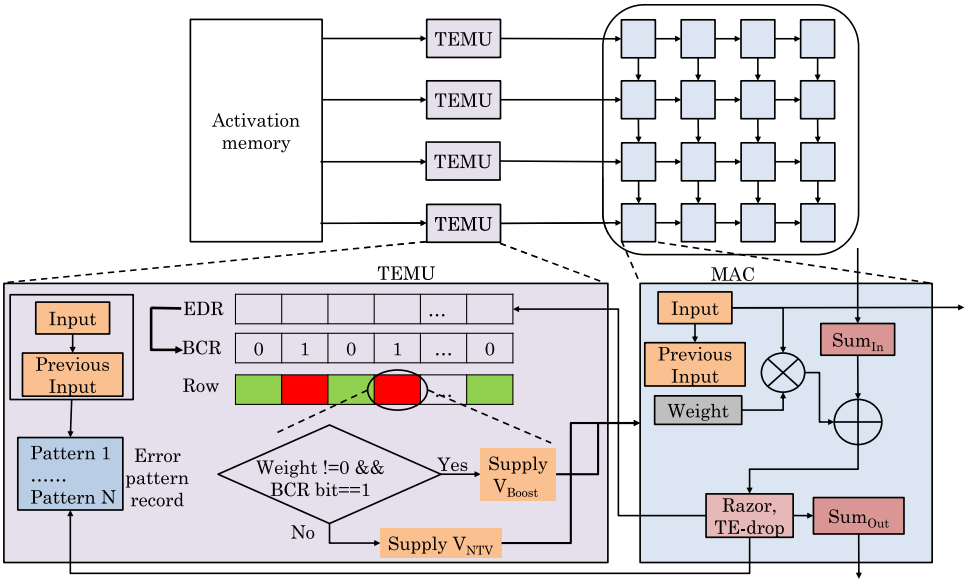


Fig. 24. The extensions to TPU proposed by Pandey et al. [48] for error detection and correction (EDR= error detection register, BCR=boost control register).

Pandey et al. [48] suggest operating Google's TPU at NTV for saving energy and propose techniques for mitigating TEs due to NTV operation. TPU has homogeneous architecture, and it performs 8-bit integer arithmetic. Hence, the number of distinct sensitized path delays seen by a MAC unit is small in number. In a MAC unit, the logic depth of the multiplier is higher than that of the accumulator. Hence, they model the delay characteristics of a PV-hit MAC at NTV by supplying all possible combinations of inputs to the multiplier, i.e., 8-bit weights and activations. They find that no path is sensitized on applying the same activation sequence in two successive cycles. Also, most multiply operations sensitize path with small delays, and only a few input patterns create TEs. This makes it easy to predict the TEs that repeat due to the same input sequence.

They observe that an input sequence that creates a TE in a MAC unit is highly likely to create TEs in later MACs also, as long as the sequence remains alive in the row. Thus, their technique predicts errors depending on the input sequences. As shown in Fig. 24, in each row of TPU, they insert a "TE management unit" (TEMU) between the activation memory and the systolic array. TEMU predicts and prevents TEs in the MAC units. A TE is detected using Razor-like flip flop. On the occurrence of a TE in a MAC unit of a row, its TEMU records the input pattern that led to the TE. Further, in the next cycle, the supply voltage of succeeding MACs in the row is increased from NTV (0.45V) to a "boost voltage" (0.65V) for preventing further TEs. Also, if a future input pattern is found to match the above-noted input pattern, the supply voltage is boosted for preventing TEs. Switching between NTV and boost voltage takes one clock cycle and also incurs energy overhead. Since a MAC unit storing a weight of zero is not likely to see a TE, its voltage is not boosted.

Since the degrees of the contribution of various bits of an input pattern to the sensitization delay are different, multiple input patterns are combined for efficiently storing them. By using the TE-drop technique [14], the impact of TE on computation is avoided. The 8-bit input pattern leading to this TE is recorded, and the fault-free output is calculated in parallel. Compared to the TE-drop technique [14], their technique significantly reduces TEs at NTV and also improves performance.

4.6. Techniques for addressing security vulnerabilities due to reliability issues

The DNN model is a valuable intellectual property (IP) for a company that needs to be protected from the adversaries since the knowledge of design topology and hyperparameters of a DNN model can allow an adversary to manipulate its operation and gain commercial advantage. Zhao et al. [5] propose a technique for achieving IP protection by leveraging the uniqueness of error-mask of DNN accelerator device, specifically its DRAM. Fig. 25 shows the working flow of their technique. During training, their technique maintains two groups of weights: inexact weights which are used for FWP and exact weights which are for weight update. They note that the strategy of applying the memory error mask at the time of generating the imprecise weights cannot maintain the inference accuracy. This is because introducing errors in the integer bits of weights leads to high errors. For mitigating these errors, they add batch normalization layer and show that this recovers the accuracy loss.

After completion of training on the server, exact weights (and not inexact weights) are sent to the NN accelerator ("target device"). The target device has the error mask, and hence, it can obtain inexact weights on loading the exact weights to DRAM. Now, assuming that the exact weights become known to an attacker, she may either use these weights on another device or may use the error mask of another device. They show that on using the exact weights, the degradation in accuracy is insignificant on the target device but very high on any other device, including a fault-free device.

To generate device-dependent error masks, they reduce tRAS (row active time) parameter, e.g., from 35 ns, it is reduced to 11ns. This leads to device-dependent restore errors due to process variation. Due to restore errors, some cells may map as 0 and others as 1. Hence, the error mask has two submasks, which denote the cells stuck at 0 and 1, respectively. They also propose strategies for protecting the error mask from the attacker. Since memory energy contributes a large fraction of NN accelerator energy, aggressive reduction in tRAS parameter leads to a large improvement in performance and energy efficiency.

Hong et al. [2] note that vulnerability of DNN parameters has crucial implications on the security of DNN. Specifically, based on their observations about the vulnerability of different bits of an FP32 value (refer Section 3.2), an adversary can attack the most vulnerable bit (31st bit) of vulnerable parameters. They also study "targeted misclassification attack" where an adversary seeks to preserve the overall accuracy of DNN but makes it wrongly classify a certain sample. The number of vulnerable parameters for this attack is less than that for causing a high loss in overall accuracy. They further show that an adversary program running in the same physical machine can cause single-bit flips using row-hammer attack and bring down the accuracy greatly without any information about the DNN.

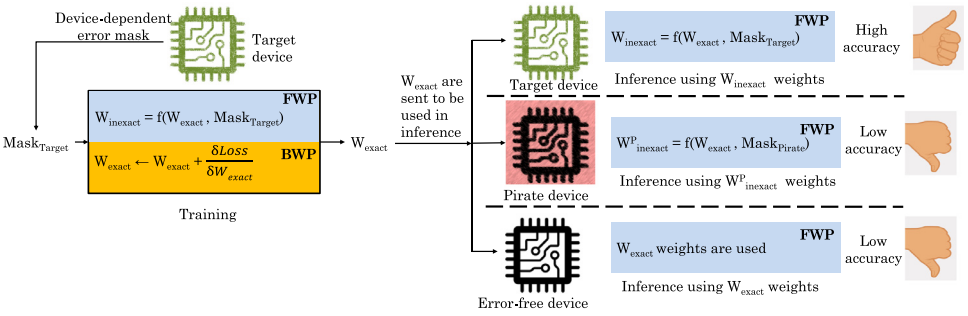


Fig. 25. Working of the technique of Zhao et al. [5] .

5. Concluding remarks

In this paper, we reviewed the techniques for modeling and improving the resilience of DNN algorithms and accelerators. We reviewed the impact of various design-decisions on the reliability and summarized the techniques for mitigating errors. We included several types of reliability issues occurring in various processor units and their components. We also discussed the implications of reliability techniques on performance, energy efficiency, and security. We end this paper with a brief mention of avenues for future research.

DNN algorithms continue to evolve at a rapid pace, and so do the DNN accelerators. As such, improving their resilience is similar to "chasing a moving target". Evidently, the design of comprehensive and efficient reliability solutions will require synergistic efforts from researchers in both AI domain and reliability domain.

Checkpointing is a commonly-used scheme to recover from errors. Modern DNNs have a large memory footprint (weights + activations), and many real-world applications require the use of multiple DNNs. For example, in autonomous driving, different DNNs are required for object tracking, localization, and segmentation. Clearly, a naive checkpointing strategy will lead to significant overhead. Novel checkpointing techniques are required, which incur low overheads by virtue of leveraging the properties of DNNs.

Conventionally, DL models have been treated as black-boxes since the internal steps they take to arrive at a decision is not well-understood. This makes it challenging to predict the impact of a fault on the final result or ascertain, which neurons/connections were responsible for a change in the final result. These challenges have led the researchers to explore "explainable deep learning". The insights gained from these studies will help significantly in improving the efficacy of reliability solutions and reducing their overheads.

Some application scenarios such as medical, defense, low-power operation and harsh environments such as space and high temperature present very strict requirement of reliability. DNNs will find adoption in these scenarios only if their reliability has been thoroughly proven. As such, these scenarios can be used as the ultimate test of the efficacy of reliability solutions.

Since even one flaw in reliability can have severe consequences, ensuring reliability needs to be given utmost importance. Moving forward, the challenge of resilience has to be addressed at all levels of the system stack. The circuit-design should account for the vulnerability profile of different components to select the optimal cell designs, memory technologies, and hardening schemes [77]. For example, non-volatile memories such as spintronic memory are immune to particle-strike induced soft errors [78], although they show soft errors due to other phenomenon [77]. At microarchitecture level, design of processor components/domains with heterogeneous reliability strength will allow mapping variables/computations of different precision, types (e.g., weights/activations) and layers based on their vulnerability. Similarly, the processing-in-memory approach can lower memory accesses to DRAM memory and thus, reduce refresh requirements and error rates [79].

At the application level, an understanding of the error-resilience and deadline-slack available in the application can allow provisioning "just right" budget of reliability. Specifying the resilience/timing-slack at the level of the programming language itself will allow the designer to convey her insights directly but will also require support from compiler and ISA designers. To avoid undue emphasis on throughput and accuracy at the cost of reliability, comprehensive evaluation procedures and unified metrics need to be developed. For example, formal verification methods can be developed for arguing about reliability of a broad range of DNNs. The policymakers need to standardize the requirements of reliability in different application contexts and enforce them strictly. Overall, reliability needs to be considered as a first-class design constraint at all levels of abstraction.

Declaration of Competing Interest

The author declares no competing interest.

References

[1] S. Mittal, A survey of techniques for approximate computing, ACM Comput. Surv. 48 (4) (2016) 62:1–62:33.
[2] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, T. Dumitraş, Terminal brain damage: exposing the graceless degradation in deep neural networks under hardware fault attacks, USENIX Security Symposium (2019) 497–514.
[3] F.F. dos Santos, L. Draghetti, L. Weigel, L. Carro, P. Navaux, P. Rech, Evaluation and Mitigation of Soft-Errors in Neural Network-Based Object Detection in Three GPU Architectures, in: Proceedings of the International Conference on Dependable Systems and Networks Workshop (DSN-W), IEEE, 2017, pp. 169–176.
[4] G. Li, S.K.S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, S.W. Keckler, Understanding error propagation in deep learning neural network (DNN) accelerators and applications, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2017, p. 8.
[5] L. Zhao, Y. Zhang, J. Yang, AEP: An error-bearing neural network accelerator for energy efficiency and model protection, in: Proceedings of the International Conference on Computer-Aided Design (ICCAD), 2017, pp. 1047–1053.
[6] A.P. Arechiga, A.J. Michaels, The robustness of modern deep learning architectures against single event upset errors, in: Proceedings of the IEEE High Performance extreme Computing Conference (HPEC), IEEE, 2018, pp. 1–6.
[7] Y. Wang, J. Deng, Y. Fang, H. Li, X. Li, Resilience-aware frequency tuning for neural-network-based approximate computing chips, IEEE Trans. Very Large Scale Integration (VLSI) Syst. 25 (10) (2017) 2736–2748.
[8] O. Temam, A defect-tolerant accelerator for emerging high-performance applications, in: Proceedings of the ISCA, 2012, pp. 356–367.
[9] K. Xing, Training for'Unstable' CNN accelerator: A Case study on FPGA, arXiv preprint arXiv:1812.01689(2018).
[10] A. Hashmi, H. Berry, O. Temam, M. Lipasti, Automatic abstraction and fault tolerance in cortical microachitectures, in: Proceedings of the ISCA, 2011, pp. 1–10.
[11] K. Jia, Z. Liu, Q. Wei, F. Qiao, X. Liu, Y. Yang, H. Fan, H. Yang, Calibrating process variation at system level with in-situ low-precision transfer learning for analog neural network processors, in: Proceedings of the Design Automation Conference, 2018, p. 12.
[12] J.J. Zhang, T. Gu, K. Basu, S. Garg, Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator, in: Proceedings of the VLSI Test Symposium (VTS), 2018 IEEE 36th, IEEE, 2018, pp. 1–6.
[13] J. Deng, Y. Rang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, et al., Retraining-based timing error mitigation for hardware neural networks, in: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015, pp. 593–596.
[14] J. Zhang, K. Rangineni, Z. Ghodsi, S. Garg, Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep neural network accelerators, arXiv preprint arXiv:1802.03806(2018).

[15] M. Sabbagh, C. Gongye, Y. Fei, Y. Wang, Evaluating fault resiliency of compressed deep neural networks, in: Proceedings of the IEEE International Conference on Embedded Software and Systems (ICESS), 2019, pp. 1–7.

[16] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S.K. Lee, N. Mulholland, D. Brooks, G.-Y. Wei, Ares: a framework for quantifying the resilience of deep neural networks, in: Proceedings of the Design Automation Conference, ACM, 2018, p. 17.

[17] A.P. Arechiga, A.J. Michaels, The effect of weight errors on neural networks, in: Proceedings of the IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), IEEE, 2018, pp. 190–196.

[18] D.-T. Nguyen, N.-M. Ho, I.-J. Chang, St-DRC: Stretchable DRAM Refresh Controller with No Parity-overhead Error Correction Scheme for Energy-efficient DNNs, in: Design Automation Conference, 2019, pp. 205:1–205:6.

[19] N. Chandramoorthy, K. Swaminathan, M. Cochet, A. Paidimarri, S. Eldridge, R. Joshi, M. Ziegler, A. Buyuktosunoglu, P. Bose, Resilient low voltage accelerators for high energy efficiency, in: Proceedings of the International Symposium on High Performance Computer Architecture (HPCA), 2019, pp. 147–158.

[20] S. Mittal, J. Vetter, A survey of techniques for modeling and improving reliability of computing systems, IEEE Trans. Parall. Distrib. Syst. (2015).

[21] S. Mittal, M.S. Inukonda, A survey of techniques for improving error-Resilience of DRAM, J. Syst. Archit. (2018).

[22] S. Mittal, A survey of reRAM-based architectures for processing-in-memory and neural networks, Mach. Learn. Knowl. Extract. 1 (2018) 5.

[23] J.J. Zhang, K. Liu, F. Khalid, M.A. Hanif, S. Rehman, T. Theocharides, A. Artussi, M. Shafique, S. Garg, Building robust machine learning systems: Current progress, research challenges, and opportunities, in: Proceedings of the ACM/IEEE Design Automation Conference (DAC), 2019, pp. 1–4.

[24] C. Torres-Huitzil, B. Girau, Fault and error tolerance in neural networks: a review, IEEE Access 5 (2017) 17322–17341.

[25] J.-S. Kim, J.-S. Yang, DRIS-3: Deep Neural Network Reliability Improvement Scheme in 3D Die-Stacked Memory Based on Fault Analysis, in: Proceedings of the Design Automation Conference (DAC), 2019, pp. 129:1–129:6.

[26] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S.S. Mukherjee, R. Rangan, Computing architectural vulnerability factors for address-based structures, in: Proceedings of the International Symposium on Computer Architecture (ISCA), 2005, pp. 532–543.

[27] C. Lunardi, F. Previlon, D. Kaeli, P. Rech, On the efficacy of ECC and the benefits of finfet transistor layout for GPU reliability, IEEE Trans. Nuclear Sci. 65 (8) (2018) 1843–1850.

[28] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Depend. Secure Comput. 1 (1) (2004) 11–33.

[29] S. Mittal, A survey of architectural techniques for near-Threshold computing, ACM J. Emerg. Technol. Comput. Syst. 12 (4) (2015) 46:1–46:26.

[30] S. Mittal, A survey of architectural techniques for managing process variation, ACM Comput. Surv. 48 (4) (2016) 54:1–54:29.

[31] J.J. Zhang, S. Garg, FATE: fast and accurate timing error prediction framework for low power DNN accelerator design, in: Proceedings of the International Conference on Computer-Aided Design, ACM, 2018, p. 24.

[32] D. Ernst, N.S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, et al., Razor: A low-power pipeline based on circuit-level timing speculation, in: Proceedings of the International Symposium on Microarchitecture, 2003, p. 7.

[33] Mathworks, (https://es.mathworks.com/help/nnet/ref/satlin.html).

[34] Mathworks, (https://es.mathworks.com/help/nnet/ref/logsig.html).

[35] S. Yin, S. Tang, X. Lin, P. Ouyang, F. Tu, J. Zhao, C. Xu, S. Li, Y. Xie, S. Wei, et al., Parana: a parallel neural architecture considering thermal problem of 3D stacked memory, IEEE Trans. Parallel Distrib. Syst. 30 (1) (2018) 146–160.

[36] X. Jiao, M. Luo, J.-H. Lin, R.K. Gupta, An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations, in: Proceedings of the International Conference on Computer-Aided Design, IEEE Press, 2017, pp. 945–950.

[37] Q. Shi, H. Omar, O. Khan, Exploiting the tradeoff between program accuracy and soft-error resiliency overhead for machine learning workloads, arXiv preprint arXiv:1707.02589(2017).

[38] T. Marty, T. Yuki, S. Derrien, Algorithm Level Timing Speculation for Convolutional Neural Network Accelerators, Univ Rennes, Inria, CNRS, IRISA, France, 2018 Ph.D. thesis.

[39] A. Azizimazreah, Y. Gu, X. Gu, L. Chen, Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs, in: Proceedings of the IEEE International Conference on Networking, Architecture and Storage (NAS), IEEE, 2018, pp. 1–10.

[40] C. Schorn, A. Guntoro, G. Ascheid, Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators, in: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, IEEE, 2018, pp. 979–984.

[41] D.T. Nguyen, H. Kim, H.-J. Lee, I.-J. Chang, An approximate memory architecture for a reduction of refresh power consumption in deep learning applications, in: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), 2018, pp. 1–5.

[42] W. Choi, D. Shin, J. Park, S. Ghosh, Sensitivity based error resilient techniques for energy efficient deep neural network accelerators, in: Proceedings of the Design Automation Conference, 2019, pp. 204:1–204:6.

[43] F.F. dos Santos, P.F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, P. Rech, Analyzing and increasing the reliability of convolutional neural networks on GPUS, IEEE Trans. Reliab. (2018).

[44] G.B. Hacene, F. Leduc-Primeau, A.B. Soussia, V. Gripon, F. Gagnon, Training modern deep neural networks for memory-fault robustness, in: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), 2019, pp. 1–5.

[45] A. Bosio, P. Bernardi, A. Ruospo, E. Sanchez, A reliability analysis of a deep neural network, in: Proceedings of the IEEE Latin American Test Symposium (LATS), 2019, pp. 1–6.

[46] C. Schorn, A. Guntoro, G. Ascheid, An efficient bit-flip resilience optimization method for deep neural networks, in: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019, pp. 1507–1512.

[47] F. Fernandes dos Santos, C. Lunardi, D. Oliveira, F. Libano, P. Rech, Reliability evaluation of mixed-precision architectures, in: Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), 2019, pp. 238–249.

[48] P. Pandey, P. Basu, K. Chakraborty, S. Roy, GreenTPU: Improving Timing Error Resilience of a Near-Threshold Tensor Processing Unit, in: Proceedings of the Design Automation Conference, 2019, p. 173.

[49] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S.K. Lee, J.M. Hernández-Lobato, G.-Y. Wei, D. Brooks, Minerva: Enabling low-power, highly-accurate deep neural network accelerators, in: Proceedings of the International Symposium on Computer Architecture (ISCA), IEEE, 2016, pp. 267–278.

[50] , MATIC: Learning around errors for efficient low-voltage neural network accelerators, author=Kim, Sung and Howe, Patrick and Moreau, Thierry and Alaghi, Armin and Ceze, Luis and Sathe, Visvesh, in: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, IEEE, 2018, pp. 1–6.

[51] B. Salami, O. Unsal, A. Cristal, On the resilience of RTL NN accelerators: fault characterization and mitigation, arXiv preprint arXiv:1806.09679(2018).

[52] F. Libano, B. Wilson, J. Anderson, M. Wirthlin, C. Cazzaniga, C. Frost, P. Rech, Selective hardening for neural networks in FPGAs, IEEE Trans. Nuclear Sci. (2018).

[53] I.C. Lopes, F. Benevenuti, F.L. Kastensmidt, A.A. Susin, P. Rech, Reliability analysis on case-study traffic sign convolutional neural network on APSoC, in: Proceedings of the 19th Latin-American Test Symposium (LATS), 2018, pp. 1–6.

[54] F. Benevenuti, F. Libano, V. Pouget, F.L. Kastensmidt, P. Rech, Comparative Analysis of Inference Errors in a Neural Network Implemented in SRAM-Based FPGA Induced by Neutron Irradiation and Fault Injection Methods, in: Proceedings of the Symposium on Integrated Circuits and Systems Design (SBCCI), 2018, pp. 1–6.

[55] H.R. Mahdiani, S.M. Fakhraie, C. Lucas, Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors, IEEE Trans. Neural Netw. Learn. Syst. 23 (8) (2012) 1215–1228.

[56] P.N. Whatmough, S.K. Lee, D. Brooks, G.-Y. Wei, DNN Engine: a 28-nm timing-Error tolerant sparse deep neural network processor for IoT applications, IEEE J. Solid-State Circu. 53 (9) (2018) 2722–2731.

[57] M.A. Neggaz, I. Alouani, P.R. Lorenzo, S. Niar, A Reliability Study on CNNs for Critical Embedded Systems, in: Proceedings of the IEEE International Conference on Computer Design (ICCD), 2018, pp. 476–479.

[58] A.L. Sartor, P.H. Becker, A.C. Beck, A fast and accurate hybrid fault injection platform for transient and permanent faults, Design Autom. Embedded Syst. (2018) 1–17.

[59] G. Srinivasan, P. Wijesinghe, S.S. Sarwar, A. Jaiswal, K. Roy, Significance driven hybrid 8t-6t sram for energy-efficient synaptic storage in artificial neural networks, in: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, pp. 151–156.

[60] L. Yang, B. Murmann, Sram voltage scaling for energy-efficient convolutional neural networks, in: Proceedings of the 18th International Symposium on Quality Electronic Design (ISQED), 2017, pp. 7–12.

[61] M. Klachko, M.R. Mahmoodi, D.B. Strukov, Improving noise tolerance of mixed-signal neural networks, arXiv preprint arXiv:1904.01705(2019).

[62] A.S. Rekhi, B. Zimmer, N. Nedovic, N. Liu, R. Venkatesan, M. Wang, B. Khailany, W.J. Dally, C.T. Gray, Analog/mixed-signal hardware error modeling for deep learning inference, in: Proceedings of the Design Automation Conference, 2019, p. 81.

[63] S. Eldridge, A. Joshi, Exploiting hidden layer modular redundancy for fault-tolerance in neural network accelerators, in: Proceedings of the Boston area ARChitecture (BARC) Workshop, 2015.

[64] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, K.-R. Müller, Explaining nonlinear classification decisions with deep taylor decomposition, Pattern Recogn. 65 (2017) 211–222.

[65] S. Mittal, S. Vaishay, A survey of techniques for optimizing deep learning on GPUs, J. Syst. Arch. (2019).

[66] S.K.S. Hari, T. Tsai, M. Stephenson, S.W. Keckler, J. Emer, SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation, in: Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS), 2017, pp. 249–258.

[67] S. Mittal, A survey on optimized implementation of deep learning models on the NVIDIA jetson platform, J. Syst. Arch. (2019).

[68] C. Schorn, A. Guntoro, G. Ascheid, Efficient on-line error detection and mitigation for deep neural network accelerators, in: Proceedings of the International Conference on Computer Safety, Reliability, and Security, Springer, 2018, pp. 205–219.

[69] E.B. Tchernev, R.G. Mulvaney, D.S. Phatak, Investigating the fault tolerance of neural networks, Neural Comput. 17 (7) (2005) 1646–1664.

[70] S. Dutta, Z. Bai, T.M. Low, P. Grover, Codenet: training large scale neural networks in presence of soft-Errors, arXiv preprint arXiv:1903.01042(2019).

[71] S. Mittal, S. Nag, A survey of encoding techniques for reducing data-Movement energy, J. Syst. Arch. 97 (2019) 373–396.

[72] Safety standard 2016. iso26262: Road vehicles functional safety, 2016, (https://en.wikipedia.org/wiki/ISO_26262).

[73] S. Mittal, J. Vetter, Reliability tradeoffs in design of volatile and non-volatile caches, J. Circu. Syst. Comput. (2016).

[74] S. Mittal, J. Vetter, A survey of techniques for architecting DRAM caches, IEEE Trans. Parall. Distrib. Systems (TPDS) (2015), doi:10.1109/TPDS.2015.2461155.
[75] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, arXiv preprint arXiv:1611.06440(2016).
[76] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, T. Mudge, A self-tuning DVS processor using delay-error detection and correction, IEEE J. Solid-State Circu. 41 (4) (2006) 792–804.
[77] S. Mittal, A survey of soft-Error mitigation techniques for non-Volatile memories, Computers 6 (8) (2017).
[78] S. Umesh, S. Mittal, A survey of spintronic architectures for processing-in-memory and neural networks, J. Syst. Archit. (2018).
[79] S. Mittal, Survey on applications and architectural-optimizations of Micron's automata processor, J. Syst. Archit. (2019).

Sparsh Mittal received the B.Tech. degree in electronics and communications engineering from IIT, Roorkee, India and the Ph.D. degree in computer engineering from Iowa State University (ISU), USA. He worked as a Post-Doctoral Research Associate at Oak Ridge National Lab (ORNL), USA for 3 years. He is currently working as an assistant professor at IIT Hyderabad, India. He was the graduating topper of his batch in B.Tech and has received fellowship from ISU and performance award from ORNL. Sparsh has published more than 80 papers in top conferences and journals. My research has been covered by several technical news websites, e.g. Phys.org, InsideHPC, Primeur Magazine, StorageSearch, Data- Compression.info, TechEnablement, ScientificComputing, SemiEngineering, ReRAM forum and HPCWire. His research interests include accelerators for neural networks, architectures for machine learning, nonvolatile memory, and GPU architectures. His webpage is http://www.iith.ac.in/~sparsh/.