Yield Loss Reduction and Test of AI and Deep Learning Accelerators

Mehdi Sadi and Ujjwal Guin
Dept. of Electrical & Computer Engineering, Auburn University, Auburn, AL, USA
Emails: {mehdi.sadi, ujjwal.guin}@auburn.edu

ABSTRACT

With data-driven analytics becoming mainstream, the global demand for dedicated AI and Deep Learning accelerator chips is soaring. These accelerators, designed with densely packed Processing Elements (PE), are especially vulnerable to the manufacturing defects and functional faults common in the advanced semiconductor process nodes resulting in significant yield loss. In this work, we demonstrate an application-driven methodology to reduce the yield loss of AI accelerators by correlating the circuit faults in the PEs of the accelerator with the desired accuracy of the AI workload execution. We exploit the error-healing properties of backpropagation during training, and the inherent fault tolerance features of trained deep learning models during inference to develop the presented yield loss reduction and test methodology. An analytical relationship is derived between fault location, fault rate, and the AI task’s accuracy for deciding if the accelerator chip can pass the final yield test. An yield-loss reduction aware fault isolation, ATPG, and test flow are presented for the multiply and accumulate units of the PEs. Results obtained with widely used AI/deep learning benchmarks demonstrate the efficacy of the proposed approach in the reduction of yield loss of AI accelerator designs while maintaining the desired accuracy of AI tasks.

I. INTRODUCTION

The demand for Artificial Intelligence (AI) and deep learning is growing at a rapid pace across a wide range of applications such as, self-driving vehicles, image and voice recognition, medical imaging and diagnosis, finance and banking, natural resource explorations, defense operations, etc. Because of these data-driven analytics and AI boom, demands in deep learning and AI will emerge at both data centers and the edge [1]-[8]. In a recent market research [8], it has been reported that AI-related semiconductors will see a growth of about 18 percent annually over the next few years - five times greater than the rate for non-AI applications. By 2025, AI-related semiconductors could account for almost 20 percent of all semiconductor demand, which would translate into about $67 billion in revenue [8].

Although GPU was adopted by the AI community, by design GPUs were not optimized for AI workloads [8][9]. As a result significant R&D efforts in developing AI accelerators - optimized to achieve much higher throughput in deep learning compared to GPUs - are underway from academia [2][3], big techs [4]-[6], as well as startups [7][10]. Dedicated accelerators are in high demand for both the cloud-based training, and inference tasks on edge devices. The training procedure is time-consuming as it requires many learning samples to adapt the network parameters. For instance, a self-driving car’s neural network has to be trained with many images of possible objects it can encounter on the road, and this will require multiple high-performance AI accelerators on the cloud. During inference, AI algorithms handle less data but rapid responses are required as they are often used in critical time-sensitive applications. For example, an autonomous vehicle has to make immediate decisions on objects it sees during driving, a medical device must interpret a trauma patient’s brain scans immediately. As a result, high throughput accelerators running on edge devices and capable of fast inference are required. In AI technology innovation and leadership, high-throughput AI accelerator hardware chips will serve as the differentiator [8][10].

Millions of Multiply and Accumulate (MAC) operations are needed in modern AI tasks, for example, AlexNet [11] and ResNET-50 [12] require 0.7 Billion and 3.7 Billion MAC operations, respectively, to classify a single image from the ImageNet [14] dataset. Modern AI accelerators contain thousands of Processing Elements (PE) distributed in densely packed arrays in a single chip/die [1]-[5] or in a multi-chiplet based chip [10]. To accommodate as many PEs as possible in the AI accelerator, a large chip (e.g., wafer-scale) would be the best solution [10]. However, manufacturing large chips is difficult due to the long interconnect wires and the possibility of particle defects. Aggressive scaling of design rules [15], lithography imperfections [16], local edge roughness, interconnect pitch reduction, etc., in 10nm and newer semiconductor technologies [15] have caused yield (i.e., the fraction of total manufactured chips that can be sold to the customer) to become as important as the conventional design metrics of Power, Performance, Area (PPA) for the process to be economically viable [17][18]. Moreover, internal cell defects have become a major yield limiter because of aggressive scaling of design rules and lithography limitations in printing the features, giving rise to cell-aware test [19]. In addition to the regular stuck-at and timing faults at the cell I/O level, at advanced technologies, the transistor-level and other cell-internal faults need to be added to the fault list as yield has become more vulnerable to internal cell defects [20]. Although yield data of semiconductor process are considered a well-guarded trade secret and not published, it is well known that yield loss has caused significant delays in product readiness, and loss in market share and revenue for a leading processor manufacturer at 10nm [21]. To integrate more PE in the accelerator, a two-level chiplet [22] based approach, where many PE are placed on a smaller chiplet, and then multiple chiplets are connected with silicon interconnect fabric [23] to form the accelerator is a viable solution. Although a relatively smaller size would minimize particle-induced random defects on upper metal layers in the chiplets and improve...
defect-induced yield loss, the individual PEs - internal to the chiplet - will still be susceptible to systematic defects and lithography imperfections that impact the transistor layers or the ultra-dense lower metal layers.

As many PEs are densely placed in the AI accelerators, defects and circuit faults are likely to occur in some PEs. Fortunately, the stochasticity inherent in the backpropagation-based training of Deep Neural Networks (NN) and Deep Convolutional Neural Networks (CNN) - the primary building blocks of AI systems - offers a certain degree of resilience and error tolerance to deep learning tasks. Moreover, the intelligent application of techniques such as dropout, pruning, and quantization during training can further increase the robustness of a trained NN/CNN against variations and noise.

Following this approach, an AI accelerator with some faulty PEs will still function correctly despite having few faulty PEs, thus demonstrating that AI accelerator chiplet/chips can still perform their logic cones into categories: (i) critical (unacceptable), (ii) non-critical (acceptable), according to their impact on the accuracy of the AI task. This relationship is later used in demonstrating that AI accelerator chiplet/chips can still function correctly despite having few faulty PEs, thus saving yield significantly.

In this paper, we propose **YOTA: Yield and Accuracy aware Optimum Test of AI accelerators**, which considers the accuracy-sensitivity and fault-tolerance of AI applications into test pattern generation for the accelerator chip/chiplet in deciding whether it will pass the yield test. The key contributions and highlights of this paper are as follows:

- An analytical relationship is established - based on the actual AI workload to be executed - between the (i) faults of the MAC modules, (ii) the rate of faults, and the accuracy of the AI task. This relationship is later used in demonstrating that AI accelerator chiplet/chips can still function correctly despite having few faulty PEs, thus saving yield significantly.
- An accuracy-aware fault isolation and test pattern generation methodology is presented to group the MAC faults by their logic cones into categories: (i) critical (unacceptable), and (ii) non-critical (acceptable), according to their impact on the accuracy of AI workload. Responses from these test patterns dictate yield decisions - whether to ship to the customer at reduced throughput, or discard as yield loss. Using the reduced test pattern set for critical faults, only PEs will require runtime hardware reconfiguration for different speed/throughput bins.
- A strategy of retraining and selective deactivation of faulty PEs during inference is presented to minimize the accuracy loss due to faulty MACs.
- Simulation results from 50,000 image samples on widely used CNN (AlexNet, ResNet-50, VGG16, LeNet5) and 10,000 data samples on different NN architectures are presented to demonstrate the relationship between fault rate and accuracy. Results show that with 5% fault rate the normalized accuracy of NN and CNN only degrade by less than 1%.

The rest of the paper is organized as follows. Related work and AI/Deep learning accelerator backgrounds are covered in Section II. The proposed YOTA methodology, test flow, and hardware control scheme are presented in Section III. Simulation results are presented in Section IV followed by conclusions in Section V.

## II. Background

### A. Related Work

The correlation between the AI task’s accuracy and computational errors in PE (i.e., multiplier and adders) has been studied by researchers in developing energy-efficient AI accelerator architectures. In [28]-[31], by exploiting the inherent error tolerance of NN/CNN to improve energy efficiency at the cost of minor accuracy loss, approximate computing (e.g., low power approximate multipliers) techniques with retraining were applied. From the backpropagation analysis for a certain benchmark, each neuron was ranked according to its sensitivity and error contribution to the output of the network, and neurons that contributed the least to the global error were then approximated [28]-[29]. To recover a portion of the loss caused by approximation, the NN/CNN was incrementally retrained with the approximations in-place. In hardware execution, neurons that were approximated were assigned to approximate PEs, while others were assigned to exact PEs [28]-[29]. In [31] errors caused by the use of approximate multipliers were modeled as noise similar to the dropout technique [22]. Retraining was performed to adapt the NN/CNN to this workload-specific noise patterns such that only a minor amount of accuracy was lost during inference because of the use of approximate multipliers. The authors in [31] demonstrated that, in addition to being workload/benchmark dependent, the success of their approach was also dependent on the hardware structure of the approximate multipliers. However, the challenges with these approaches are, (i) sensitivity-based sorting, approximation, and retraining of neurons are always dependent on the workload [31]. For example, the same network will have different sensitivities for SVHN and ImageNet datasets. (ii) Assigning less sensitive neurons to approximate PEs will require runtime hardware reconfiguration for different tasks. (iii) In [28]-[31] pruning was not considered, but in modern NN/CNN pruning is applied as a well-established method to reduce network size where less important/sensitive connections are already pruned/removed during training [24]-[26]. Hence, the techniques of [28]-[31] will be much less effective when applied on an already pruned NN/CNN. (iv) Since these approximate CNN/NN assume that the only source...
of error is the deterministic approximate multiplier and adders. any additional faults in the execution hardware from process variation induced defects will cause a significant amount of inaccuracy in the prediction during inference, and will further require costly approximation-aware fault testing [33][34].

With the widespread use of NN/CNN and the rise of AI accelerators, the testing of these hardware has become an emerging research problem [32][34]. In [32], a comprehensive structural test flow was proposed that first identified critical faults by comparing the accuracies of the exact fault-free gate-level circuit of the neural network, and that of a faulty version. Next, the entire circuit was converted into an Boolean satisfiability (SAT) instance and solved with SAT solver with test patterns for the critical faults only. Since this approach is expensive and not scalable, a functional test method was proposed [32], where real workloads - the test images from MNIST and CIFAR10 benchmarks - are applied as test input to the gate-level netlist of the implemented neural network. To find if a fault was critical or ignorable, the fault was first injected into a neuron module and the test image set was applied. If the prediction accuracy of the test image set was within a certain threshold, the fault was considered unimportant, otherwise, it was a critical fault. The proposed simple approach of creating RTL of the full neural network, followed by gate-level synthesis and fault injection may be applicable for the basic neural networks classifying images with at most 10 possible classes (MNIST and CIFAR10 as used in [32]), however, this approach will not be scalable for mainstream CNNs [29] such as AlexNet, VGG, ResNet, MobileNet, etc. that are used for real-world computer vision tasks of image recognition with many classes [1][2]. For instance, the widely used ImageNet [14] benchmark has 1000 classes. Moreover, in contrast to the very basic neural networks such as MNIST/CIFAR10 classifiers, the standard CNNs require millions of Multiply and Accumulate (MAC) operations to classify a single image [39].

B. AI/Deep Learning Accelerator Architecture and Processing Element Faults

![Deep Neural Network (NN)](image1)

(a) Deep Neural Network (NN)

![Deep Convolutional Neural Network (CNN)](image2)

(b) Deep Convolutional Neural Network (CNN)

(c) Multiply and accumulate (MAC) in convolution operation

Figure 1. (a) Deep NN; (b) Deep CNN; (c) Convolution in CNN [1][29].

At the essence of AI and Deep Learning Convolution algorithms are the backpropagation-based NN and CNN. As shown in Fig. 1(a), a deep NN consists of an input layer, followed by several hidden layers and one final output layer. Depending on the size of training data samples, the complexity of training, dropout, and pruning rate, some layers in the NN can be fully connected and others sparsely connected [23][27]. The connection strengths between the adjacent layers are represented by a weight matrix \( W \), and the matrix parameters \( w_i \) are learned by the backpropagation equation, \( \Delta w_i = -a \cdot \frac{\partial E}{\partial w_i} \), where \( a \) is learning rate and \( E \) is the prediction error. During the forward pass at both training and inference phases, the output activation of a layer, \( X_n \), is obtained by multiplying the input activation vector with the weight matrix followed by addition of a bias term (if any), and finally passing the result through an non-linear function such as ReLU, \( X_n = ReLU(X_n * W + b) \). It is evident from this equation that the dominant operation in NN is Multiply & Accumulate (MAC) in matrix multiplications and bias term additions, of which multiplication is the most hardware intensive. For a fully connected NN with input layer of size \( N_{input} \), \( K_h \) hidden layers each of size \( N_{hidden} \), and output layer of size \( N_{output} \), the total number of required multiplications to classify a single sample is, \((N_{input} * N_{hidden}) + (N_{hidden} * K_h) + (K_h - 1) + (N_{hidden} * N_{output})\).

Because of their robustness and accuracy, deep CNNs have become the standard for image and pattern recognition [1][39]. The operation of a deep CNN is briefly shown in Fig. 1(b). During training and inference, each image or pattern is convolved successively with a set of filters where each filter has a set of kernels. After ReLU activation and pooling (i.e., subsampling), the convolution operation is repeated with a new set of filters. Finally, before the output stage, fully connected NNs are used. The convolution operation is shown in Fig. 1(c) and it consists of dot products between the input feature-maps/image and filter weights, mathematically, \( f_{out}(m,n) = \sum_j \sum_k h(j,k)f_{in}(m-j,n-k) \). For a single convolution layer, the total number of multiplications is given by, \( N_{in\_channel} * D_k * D_k * J_i * D_f * N_{out\_channel} \), where \( D_k \) is kernel dimension, \( D_f \) is output feature map dimension, \( N_{in\_channel} \) is number of input channels, and number of output channels is \( N_{out\_channel} \). For deep CNNs with many convolution layers, the total number required multiplications to classify each input image/pattern sample is a substantial number, and MAC operations in the convolution layers and fully connected layers account for 90% or more of the total computational cost [1][4].

![Processing Element (PE)](image3)

Figure 2. AI accelerator with PE arrays performing MAC.

Since the computations in NN/CNN are mostly dominated by MAC operations, the AI accelerators, as shown in Fig. 2, are primarily occupied with arrays of Processing Elements (PE) optimized for fast MAC function [1][4]. Depending on
if the accelerators are for training and inference or only for inference on edge devices, the MAC units can be of 32 bits supporting FP32 or 8 bits for int8 36 quantized operations 10 35. To optimize silicon utilization, the large number of PEs are densely placed on the chip die 11-4. As a result, the yield of the accelerator will be primarily dictated by circuit faults (e.g., stuck-at, timing, etc.) in MAC modules of the PEs. Categorization of the location and rate of the faults as critical or non-critical with respect to the desired accuracy of the AI task, and corresponding fault-criticality-aware test pattern generation can save time and cost associated with final yield evaluation. Also, this will allow more chips to pass the functional test (within an accuracy bound of the AI task) resulting in improved yield (i.e., more accelerator chip/chiplets passing the quality test at different specs) at advanced technology nodes.

III. YAOA: Yield and Accuracy Aware Optimum Test for AI Accelerators

The semiconductor manufacturing defects and corresponding circuit faults in the MAC units of the PE in an AI accelerator need a deeper investigation with regards to its impact on the accuracy of AI and deep learning workloads. Any circuit fault (e.g., stuck-at, timing, etc.) on the 32-bit (FP32 format) or 8-bit (for int8 quantization 36) multipliers and adders will cause a certain precision loss and inaccuracy at the output of MAC. The important question is, "in an AI accelerator with thousands of PEs with MAC units, will the presence of a few faulty MACs cause the whole accelerator chip to be discarded, resulting in the loss of yield and revenue?" A scientifically pragmatic solution would be to assess the impact of MAC circuit faults on the training and inference accuracy of AI workloads executed on these accelerators. The key factors to consider in this assessment are, (i) location of the faults inside the PE and its impact on the precision of MAC output, (ii) the fraction of the total PEs that have faulty MACs, (iii) the type of AI workload, and (iv) if the accelerator is for both training and inference, or inference-only.

A. Fault location, MAC Precision and AI Accuracy

Systematic defects and yield losses are caused by layout-sensitive lithographic hotspots and other process imperfections, variations, and are generally independent of the layout area 17. On the other hand, random defect generated yield losses are caused by defect particles and are dependent on the standard-cell or the layout area as well as the defect particle size 16,18. These defects (e.g., short/open defect, poor contact/via, etc.) and corresponding circuit faults can occur at different sites inside the MAC circuit block. The precision loss - due to the presence of circuit faults - at the output of multiply and accumulate operation will depend on the location of the fault inside MAC circuit and the logic cone impacted by the fault. For example, if a multiplier circuit that performs the multiplication of two 8-bit numbers, has faults impacting up to K LSB bits, then it will sustain worst-case error of $\pm 2^{K+1} \cdot 2^i$ (the last $2^{K+1}$ term comes from worst-case carry-in path of partial product addition, as explained in the next subsection). As $K$ increases, the faults impact the more significant digits causing the worst-case error of the multiplier to increase. Similarly, errors will also occur in the adder circuit of the MAC if it is corrupted by faults.

Since the multiplier is the more dominant block in a MAC, it will be more prone to faults and computation errors.

As explained in Section II(B), the computations in NNs and CNNs for an AI workload execution are heavily dominated by the MAC operations. Any inaccuracy in MAC output will impact the accuracy and efficacy of the AI task running on the accelerator. From the matrix multiplications in NN and convolution operations in CNN, it can be inferred that despite a certain amount of error in a few MAC modules, the AI tasks can be accomplished with minimum accuracy loss. The relationship between worst-case error per faulty MAC, percentage of MACs that are faulty and the AI workload’s accuracy loss due to these errors are correlated, and will depend on the specifics of the NN/CNN architecture and the workload as will be demonstrated in Section IV.

B. Yield and Accuracy Aware Fault Isolation and Test Pattern Generation

![Figure 3. Simplified block diagram of MAC unit with logic cones.](image)

Algorithm 1 Isolate critical and non-critical faults, and generate test patterns accordingly for Multiplier and Adder

1: procedure GENERATE TEST PATTERNS FOR CRITICAL AND NON-CRITICAL FAULTS
2: Input: Gate-level netlist of the MAC module
3: Input: Maximum LSB bit position of MAC output that can tolerate errors, $K$
4: Output: List of critical faults, $F_{crit}$
5: Output: List of non-critical faults, $F_{non-crit}$
6: Output: ATPG test patterns for critical faults, $T_{crit}$
7: Output: ATPG test patterns for non-critical faults, $T_{non-crit}$
8: Initialization: $G_1 = \{\}$
9: Initialization: $G_2 = \{\}$
10: for $i = 0$ to max. LSB position $K$ do
11:     $g_i$ ← all gates in the fan-in logic cone of output LSB of position $i$
12:     $G_1 ← G_1 \cup g_i$
13: end for
14: for $i = K + 1$ to MSB position $N$ do
15:     $g_i$ ← all gates in the fan-in logic cone of output bit of position $i$
16:     $G_2 ← G_2 \cup g_i$
17: end for
18: $G_{crit} = G_1 - G_2$
19: $G_{non-crit} = G_2 - G_1$
20: $\text{list of all faults (e.g., stuck-at)}$ for gates in $G_{crit}$
21: $\text{list of all faults (e.g., stuck-at)}$ for gates in $G_{non-crit}$
22: $T_{crit} ← \text{ATPG tool generated test patterns for faults in list } F_{crit}$
23: $T_{non-crit} ← \text{ATPG tool generated test patterns for faults in list } F_{non-crit}$
24: end procedure

The precision loss and the extent of computational error in a MAC will depend on the output bit positions that were corrupted by the circuit faults (e.g., stuck-at and delay). In this paper, we focus on stuck-at faults as they are more destructive compared to delay faults. By analyzing the fan-in logic cone of an output bit, we can isolate the circuit paths and standard-cell logic gates that can contribute to a stuck-at fault at that output bit. This can be explained with the multiplier and adder circuit of Fig. 3, and Algorithm 1. In Fig. 3 the green (shaded) logic
cone consists of all the logic gates that are in the fan-in cone of the first \(K\) Least Significant Bits (LSB) of the output. The red cone contains the standard-cells that are present in the fan-in logic cone of output bits at positions \(K+1\) to the MSB bit \(N\), where \(N > K\). From an extensive execution of AI benchmarks, we identify the acceptable error range of a faulty MAC and the fault rate that will not cause the AI task’s accuracy loss to exceed a threshold. We define output bit up to \(K\) LSBs to be non-critical from this workload-driven analysis. If circuit faults located within the logic cone of the first \(K\) output bits are non-critical then the resulting worst-case error of the MAC is \(\pm \sum_{i=0}^{K-1} 2^i\), because each bit position \(i\) depending on whether stuck-at-1 or stuck-at-0 - will introduce error \(\pm 2^i\), and an additional worst-case error of \(\pm 2^{K+1}\) will occur because of possibly wrong carry propagation to output bit position \(K+1\) from bit \(K\). The flow to isolate the critical and non-critical faults of a MAC for AI tasks, and corresponding ATPG pattern generation is presented in Algorithm 1. The inputs to the algorithm are the gate-level netlist of the MAC and the bit position \(K\) up to which the error is acceptable. The outputs of Algorithm 1 in Lines 4 to 7, are the lists of critical (\(T_{\text{crit}}\)) and non-critical (\(T_{\text{non-crit}}\)) faults, and the ATPG patterns (\(T_{\text{crit}}, T_{\text{non-crit}}\)) that can detect these faults. In Lines 10 to 13, for each bit position from 0 to \(K\), the standard-cell logic gates in the logic cone of that bit are identified and added to the list \(G_i\). Similarly, in Lines 14 to 17 the logic gates in the fan-in cone of the rest of the bits, \(K+1\) to \(N\) (MSB), are obtained in the list \(G_2\). In Line 18, the list \(G_2\) is subtracted from \(G_1\) to obtain the list of non-critical gate list \(G_{\text{non-crit}}\). In Lines 18 to 19, the gates in the carry-in path of bit \(K+1\) are identified and subtracted from \(G_2\) to obtain the list of error critical gates \(G_{\text{crit}}\). Finally, in Lines 21 to 24, the fault lists for these gates - \(F_{\text{crit}}\) and \(F_{\text{non-crit}}\) - are fed to the ATPG tool to obtain the test pattern sets \(T_{\text{crit}}\) and \(T_{\text{non-crit}}\) to test the AI-accuracy critical and non-critical faults, respectively.

The standard data type used in the AI domain to represent the NN connections, CNN filter weights, and activation outputs are the 8-bit int8 quantized format or 32-bit floating-point (FP32) format \([1][3][4][35]\). FP32 can be used for both training and inference, but for inference-only devices (e.g., mobile and other edge devices) the training is generally done in FP32 and then the learned parameters are scaled to int8 and loaded in these devices \([3][5]\). The Baugh-Wooley multiplier \([40]\) - widely used for multiplying two 8-bit signed numbers - is shown in Fig. 4 (a). If acceptable faults are only allowed to affect the two LSBs (i.e., \(K = 1\) in Algorithm 1), all circuits in logic-cones of outputs \(P_0\) and \(P_1\) - highlighted with red dots in Fig. 4 (a) - are considered non-critical, and the rest of the circuits are fault-critical. Similarly, in Fig. 4 (b) for the carry Look-ahead adder of the MAC, the logic cones containing the non-critical circuits and faults are shown for \(K = 1\). For AI accelerators using FP32 format floating-point multiplier and adder, as shown in Fig. 5, circuit faults in logic cones of some of the lower order (i.e., LSBs) bits of mantissa can be considered as non-critical, and the circuit faults present in the logic cones of the rest of the bits of mantissa, the exponent and sign bits need to be considered critical as they can introduce large errors in the result. The number of bits in mantissa that can be considered non-critical will depend on the amount of error introduced by stuck-at faults on those bits and the impact of this error in AI task’s accuracy loss.

Next, we propose the test flow to identify faulty PEs, and the use of a low area-overhead register memory to store the IDs of the faulty PEs. A control scheme is presented to deactivate some of the faulty PEs during inference and test, if required.

**Figure 5.** Logic cones of critical and non-critical bits are shown for 32 bit, (a) floating point multiplication; (b) floating point addition.

**Figure 6.** Proposed Failure Status Register (FSR) and controls to deactivate faulty PEs in the accelerator as needed.

**C. Post-fab Test of the AI Accelerator**

After manufacturing, the accelerator chip/die is subjected to structural and functional tests to identify if the PEs in
the densely packed arrays are functionally correct. Because of defects from imperfections in the semiconductor manufacturing process, some of the PEs will have faulty MAC units. First, the test patterns $T_{crit}$ obtained using Algorithm 1, are applied in a broadcast mode to all the PE/MACs to identify which units will introduce large errors during matrix multiplications or dot products in convolutions. The IDs of these faulty PEs are recorded in the list $Fail_ID_{crit}$. Next, the test patterns $T_{non-crit}$ are applied parallelly to all the PEs that passed the first test. The IDs of the PEs that failed this second test are recorded in the list $Fail_ID_{non-crit}$. Since the PEs that belong to $Fail_ID_{crit}$ have large MAC errors, they can be permanently disabled by the manufacturer (similar to the practice of disabling faulty cores in many-core processors [38]), or their $Fail_ID_{crit}$ data can be written to an on-chip non-volatile memory - Fault Status Register (FSR) shown in Fig. 6 - for the customer to decide if they want to use those PEs or not. The PEs belonging to $Fail_ID_{non-crit}$ have relatively small MAC errors and can be acceptable depending on, (i) how many such errors exist (i.e., number of elements in $Fail_ID_{non-crit}$), (ii) the type of AI workload that will be executed by the customer and their accuracy tolerance limits. Hence, the $Fail_ID_{non-crit}$ contents are written into the on-chip FSR for the customer to disable a fraction of these faulty PEs with software at runtime and still accomplish the AI tasks with minimal accuracy loss. As shown in Fig. 6, control signals are transmitted from the FSR to all the PEs to selectively disable the faulty PEs when needed. By adopting this scheme, the manufacturer can avoid discarding the full accelerator chip/die only because of the presence of few PEs with faulty MACs, and thereby increase yield. The overhead in this yield loss reduction are the extra on-chip register (FSR) to store the IDs and the control signal routes to disable faulty PEs.

1) Accelerator used for Inference Only: For large datasets, the training of deep NN/CNN is computationally intensive and very time consuming, and generally requires many accelerators or CPU/GPUs working in parallel. For example, training the popular CNN model of AlexNet on imageNet [14] dataset required 6 days with 2 GPUs [11]. Since mobile and other edge devices cannot sustain this computational overhead, the general trend for these devices is to use a pre-trained model during inference. In this mode, using many CPU/GPUs and dedicated accelerators the AI model is trained on the cloud, and after training the model parameters and weights are loaded in the edge device where a local AI accelerator is used to perform the MAC operations required for inference [2].

Our proposed fault-aware cloud-trained edge-inferred inference flow is shown in Fig. 7. In this approach, after the model has been completely trained in the cloud with high-performance computing, an additional analysis is performed to obtain the impact of MAC errors on inference accuracy. This can be achieved by injecting on the post-trained model in the cloud the same MAC errors that would occur in the inference accelerator of the edge device, and obtaining the corresponding accuracy changes as a look-up table of fault rate vs. accuracy changes. In summary, the proposed flow will not only generate the trained parameters of the AI model - similar to the regular cloud-training and edge-inference paradigm [1] - but also report the maximum allowed fault rate $FR_{max_non-crit}$ to be subsequently loaded into the accelerator hardware of the edge device as a bitmask. During inference, the mobile/edge accelerator’s FSR and control unit reads the $FR_{max_non-crit}$ and if it is lower than the current fault rate $FR_{non-crit}$, then the control unit sends deactivation signals to disable some of the PEs such that the fault rate is reduced to $FR_{max_non-crit}$ as shown in Fig. 7.

![Figure 7](image-url)  
**Figure 7.** Accelerator used for Inference on edge/mobile devices with cloud-trained parameters.

![Figure 8](image-url)  
**Figure 8.** Accelerator used in both Training and Inference.

### Algorithm 2 Fault-aware training and inference on AI accelerator

1. **procedure** IDENTIFY THE MAXIMUM ACCEPTABLE FAULT RATE OF PROCESSING ELEMENTS (PE) IN THE ACCELERATOR FOR A GIVEN ACCURACY TARGET
2. **Input:** Minimum acceptable accuracy for inference, $Acc_{Inf_threshold}$
3. **Input:** Training dataset for the deep CNN/NIN
4. **Input:** Fault rate (i.e., rate of faulty PEs) of the accelerator, $FR_{non-crit}$
5. **Input:** PEs of all faulty PEs in the accelerator, $F_ID = \{ Fail_ID_{non-crit}, Fail_ID_{crit} \}$
6. **Input:** Fault rate adjustment step size, $\Delta_{step}$
7. **Output:** Maximum allowed fault rate in the AI accelerator for inference at desired accuracy, $FR_{max_non-crit}$
8. **Initialization:** Training accuracy, $Acc_{Train} = 0$
9. **Initialization:** Deactivate all faulty PEs in $F_ID$ during training mode
10. **Initialization:** $FR_{max_non-crit} = FR_{non-crit}$
11. **while** ($Acc_{Train} < Acc_{Inf_threshold}$) **do**
12. \[ Acc_{Train} \leftarrow \text{training accuracy for the given dataset with } FR_{non-crit} \text{ modeled in the backpropagation} \]
13. \[ \text{if } (Acc_{Train} < Acc_{Inf_threshold}) \text{ then} \]
14. \[ FR_{max_non-crit} = FR_{max_non-crit} - \Delta_{step} \]
15. **end if**
16. **end while**
17. **end procedure**

2) Accelerator used in Training and Inference: For cases where the accelerator will be used for both training and
inference, further improvement in accuracy degradation caused by MAC errors can be accomplished with fault-aware retraining. Retraining is a popular tool in deep learning, and primarily used to reduce the size of the NN/CNN by pruning [24]-[26]. In retraining, the forward pass and backward passes are made aware of the changes in the network, and this directs Stochastic Gradient Descent based backpropagation flow to evolve the weights of the NN/CNN accordingly to minimize any accuracy loss stemming from these changes [25] [28]-[29].

The proposed methodology is shown in Fig. 8 and explained in Algorithm 2. The presence of possible errors in MAC calculations during inference is modeled mathematically and fed to the backpropagation weight-update flows similar to [28]-[31]. During the training phase, first, the control unit reads the FSR to obtain the IDs of faulty PEs. To ensure that the CNN/NN will be trained to achieve the best accuracy, all faulty PEs (both critical and non-critical) are disabled during the training phase as shown in Fig. 8 and Line 9 in Algorithm 2. In Algorithm 2 the fault-aware AI training and inference flow is shown for the accelerator. The minimum acceptable inference accuracy (Acc_Inference_Threshold), the training data set, accelerator’s non-critical fault rate (FR_non_crit), IDs of all faulty PEs and a fault adjustment step size (δ_step) are provided as inputs to the algorithm in Lines 2 to 6. The algorithm reports the maximum allowed faulty PE rate (i.e., the fraction of the total PEs in the accelerator that are faulty), FR_max_non-crit that will allow the achievement of desired inference accuracy Acc_Inference_Threshold.

In Lines 11 to 16, after each iteration of retraining - with fault effect modeled in the backpropagation - the obtained accuracy is compared with the desired inference accuracy Acc_Inference_Threshold. If the accuracy goal is not met, the fault rate FR_max_non-crit is reduced by a small step δ_step, until the desired accuracy goal is met. After the training converges and the inference accuracy target is reached, the trained weight and bias parameters of the NN/CNN are obtained. Additionally, the maximum allowed faulty PE rate, FR_max_non-crit, is reported. During inference, the control unit and FSR will read this FR_max_non-crit and disable some faulty PE to achieve this FR_max_non-crit rate.

The benefits from this combined training-inference methodology are, (i) in achieving the same inference accuracy, the fault-aware training approach will allow the accelerator to endure a higher FR_max_non-crit compared to the scenario where no retraining is used. This will allow the availability of more PEs during inference if fault-aware retraining was used. (ii) Although there will be additional time required for the retraining step, this extra cost will be amortized on the multiple inference runs. Because, the general trend in AI is to train the NN/CNN once accurately, and then this pre-trained model is used many times during inference. As a result, the extra PEs - made available by the fault-aware training - present during each inference will significantly speed up the inference tasks.

### IV. RESULTS AND ANALYSIS

To demonstrate the impact of circuit faults in the MAC unit on the accuracy of AI tasks we used the RTL of the MAC unit present in each PE. In identifying the critical and non-critical circuit faults in the MAC of a PE and corresponding test pattern generation, we analyzed both int8 and FP32 format implementations. First, the RTL of signed 8-bit MAC unit was developed, followed by gate-level synthesis with SAED [41] 28nm standard cell library with Synopsys Design Compiler (DC) [41]. During synthesis, DC tool implemented the signed multiplier using the Baugh-Wooley architecture available in DesignWare [41] IP library. The 16-bit adder unit was implemented using the Carry Look Ahead (CLA) structure from the DesignWare IP library. After the complete gate-level netlist of the MAC was available, a custom TCL script was developed for the logic cone analysis of the output bits. Without loss of generality, we selected the first two bits of the LSB (i.e., bits 0 and bit 1) as non-critical, implying that the worst-case total MAC error - due to stuck-at faults - resulting from these two bits is $\pm (2^0 + 2^1 + 2^2)$. Bit positions 2 to 7 were taken as critical. Next, using the developed TCL script with DC we obtained critical (G_crit) and non-critical (G_non-crit) standard cell logic gates present in the logic cones of critical and non-critical bits, respectively, as described in Algorithm 1 in Section 3(B). Next, the gate-level netlist and the critical and non-critical fault lists were taken to TestMAX ATPG tool [41] and three sets of ATPG test patterns were generated âÁ$ (i) with all faults, (ii) only with the critical faults, (iii) considering only the non-critical faults. The results are shown in Table I.

The reason that the number of test patterns for the all-faults case is lower than the sum of critical-only and non-critical-only cases is due to the method of ATPG pattern generation, where a single pattern can sometimes detect faults from both critical and non-critical groups. However, the test pattern counts to test the critical-only faults is less than the all-faults case in Table I. This critical pattern set can be applied first to all the PEs in a broadcast manner to identify if there are any PEs that must be disabled, this is because, having a critical fault in MAC introduces a large magnitude of error. Next, for the PEs that passed the first test, we apply the test patterns from Row 4 of Table I to test the presence of any non-critical faults. If faults are detected by this pattern set, the IDs of the faulty PEs are recorded in FSR memory as explained in Section III(c).

As discusses in Section 2, for training NN/CNN 32-bit floating-point numbers are used for best precision [1]. To isolate
the critical and non-critical faults of a 32-bit floating-point MAC, we obtained a 32-bit floating-point MAC benchmark circuit from OpenCores [42]. Similar to the int8 case above, we synthesized a gate-level netlist of the 32-bit MAC using the DesignWare IP library. For the floating-point MAC, we took the first 5 LSB bits (bits 0 to 4) of the mantissa as non-critical, and the rest of the bits of mantissa, the exponent and sign bit are considered critical. We chose these bits because FP32 precision changes by a small fraction for lower-order bits of mantissa, whereas any errors in higher-order bits of mantissa and exponent/sign bits will corrupt the MAC output. After identifying the critical and non-critical gates and corresponding faults, the fault lists and the gate-level netlist were taken to the ATPG tool and test patterns were generated similar to the int8 case above. The results are shown in Table II. First, the test patterns from Row 3 of Table II are applied to identify all faulty PEs that must be disabled to prevent significant accuracy loss in AI tasks. After that, the non-critical faults are identified with the patterns from Row 4 of Table II. All PEs that failed this second test have non-critical faults and their IDs are recorded in FSR.

To analyze the impact of PE/MAC faults on the inference accuracy of NN, we implemented a 4-layer NN with two hidden layers, and varied the number of neurons in the hidden layers. Also, both un-pruned and 30% pruned (with-retraining) [24]-[26] versions were implemented. For NN experiments we used MATLAB deep learning toolbox [43]. All weights and activations were quantized in int8 format using MATLAB Fixed-Point tool [43]. To incorporate the worst-case non-critical MAC faults - obtained from the gate-level netlist above - into the NN inference task, the matrix multiplication function in forward pass of the NN used in inference was modified in MATLAB to inject faults. The relationship between accuracy and fault rates is shown in Fig. 9. It can be observed from Fig. 9(c) that, other than the smaller 100 hidden layer case, the rest of the NN implementations are robust to faults in the MAC, with normalized accuracy changes less than 0.5% at 5% fault rate. With pruning (Fig. 9(d)), the accuracy degrades slightly more with faults. This is because with pruning less number of neurons are present, and those that are present become more important.

To assess the impact of MAC circuit faults on the accuracy of CNN, we used several key benchmark CNNs âÁŠ AlexNet [11], VGG-16 [13], ResNet-50 [12] and LeNet-5 [39]. The number of convolution, linear layers and the total number of multiplication and additions required (without pruning) to classify each image in these networks are tabulated in Table III. These results were obtained using custom functions developed in Pytorch [44]. For the smaller CNN, LeNet-5, we performed both training and inference with MNIST dataset of size 70,000 images. For the complex architectures - AlexNet, VGG-16 and ResNet-50 - training takes several days and requires multiple GPUs [11]-[13]. In Pytorch library, pre-trained versions of these CNNs are available where they were already trained with ImageNet [14] dataset having 14 million training images and 1000 possible classes. In our experiments we used these pre-trained models and performed inference with the 50,000 images from the ImageNet validation dataset. During inference, the models were quantized into int8 format using Pytorchâ¬Ás â¬ register_forward_hook â¬ library. To incorporate the worst-case non-critical MAC faults - obtained from the gate-level netlist above in the inference function of the CNN, we used Pytorchâ¬Ás â¬ register_forward_hook â¬ feature to access the data in Conv2d function and injected the MAC faults. The accuracy changes - in the standard Top-1 and Top-5 format - with MAC faults are shown in Fig. 10 for 50,000 test images from ImageNet [14]. Top-1 accuracy implies that the predicted class matches exactly the actual class (out of 1000 possible classes) and Top-5 refers to the case where the actual class is within the top 5 predicted classes [11]-[13]. From Fig. 10(e), it can be seen that the normalized accuracy in Top-1 category changes by less than 1.5% for all networks when fault rates are within 5%. For the Top-5 category in Fig. 10(f), except for the computationally intensive VGG-16 network, the normalized accuracy degradation was confined within 1% for fault rates up to 5%. Next, we pruned 30% of the filter weights of the convolution layers and repeated our fault injection experiments. Form Fig. 10(g)-(h), it can be seen that, with pruning, the normalized Top-1 accuracy degraded by a small amount with worst-case happening for ResNet-50 where it degraded by 2.2% for fault rate 5%. Even with pruning, the Top-5 normalized accuracy degradation was within 1% for fault
rates up to 5%. Note that, during our pruning experiments on AlexNet/VGG-16/ResNet-50 retraining was not done, because it would have required us to retrain the networks with 14 million images using a large number of GPUs. Retraining during pruning would improve the accuracy further [24]-[26].

As discussed in Section III(c)(2), using our proposed fault-aware retraining flow some of the accuracy loss due to faults in MAC units can be recovered by incorporating the fault effects in the backpropagation-based weight update segment and allowing the CNN to adapt accordingly. To experimentally demonstrate this technique, we used the LeNet-5 CNN architecture. We picked the simpler LeNet-5 architecture over AlexNet/VGG-16/ResNet-50 because of the computational complexity of retraining. Whereas AlexNet/VGG-16/ResNet-50 would require multiple GPUs and several days of training with ImageNet data [11]-[13], the LeNet-5 can be trained in several minutes on MNIST dataset using CPU. We used 6-core Intel core i7 CPU with 24GB RAM in this retraining experiment. Using 32-bit floating-point (FP32) format, and taking last 5 LSB bits (bits 0 to bit 4) of mantissa as non-critical, we modeled the equivalent worst-case MAC error - corresponding to faults occurring in the logic cones of these bits - in the backpropagation segment using Pytorchâ ˘A ´Z feature 

In Section II(A), we explained that the presence of defect-induced circuit faults in approximate NN/CNN will deteriorate the AI task’s accuracy as errors from approximations in MAC were already introduced in the model, and any further error from circuit faults will be detrimental. In Fig. 12 the normalized change of accuracy with respect to faults for LeNet-5 on MNIST dataset is shown for cases where exact and approximate multipliers were used in the convolution function. In [31] a comprehensive analysis (with approximation aware retraining) of different types approximate multipliers on the efficiency of neural networks were performed, and the best approximate multipliers in terms of prediction accuracy were reported. Based on these findings [31], in our experiment as an approximate multiplier, we chose mul8_134 from the open-source library of EvoApprox8b [37]. As discussed in [28]-[31], the accuracy of CNN/NN using approximate multipliers are very sensitive to proper training compared to regular multipliers, and the backpropagation training phase must be updated to account for approximate computing. In our experiment, the initial training phase was updated (using Pytorch’s hook functions [44]) to
account for the use of 8-bit approximate multiplier, also int8 quantization was used. From Fig. [12] it can be observed that the presence of faults in the 2 LSB bits will degrade the performance of neural networks with approximate multipliers significantly, compared to their counterparts that are using exact multipliers. Hence, if yield loss reduction is the primary goal, exact MAC units need to be used to account for possible circuit faults.

From these detailed analyses of gate-level synthesis, fault isolation, ATPG pattern generation for the MAC circuit, and corresponding simulation of fault effects on standard NN/CNN benchmarks, it can be observed that certain circuit faults - based on their locations in the circuit - have minimal impact on the AI task’s accuracy when the fault rate is within an upper limit. For example, with 5% fault rate in non-critical gates, the normalized Top-5 accuracy loss in CNNs is less than 1% (Fig. 10(f)(h)). If this 1% accuracy loss is acceptable, and if there are more than 5% faulty PEs, then we can deactivate some faulty PEs such that the fault rate is within 5%. As a result, an AI accelerator chip with few faulty PEs can be shipped, improving valuable yield and revenue. The tradeoff in this yield saving would be the lower number of PE blocks in the accelerator due to the deactivation of few faulty PEs to keep the fault rate within an acceptable limit (i.e., 5%), however, this will not have any functional impact, and will only reduce the throughput marginally. Furthermore, the reduced throughput PEs can be binned and priced differently without totally discarding the chip, thus saving yield.

V. CONCLUSIONS

In this paper, we presented a yield loss reduction and test methodology for AI accelerator chips densely packed with PEs. Exploiting the error-healing properties of backpropagation during training and the inherent fault tolerance features of trained AI models during inference, we obtained an analytical relationship between fault location and fault rate of MAC, and the AI task’s accuracy to guide yield decisions. Simulation results on NN/CNN show that the proposed YAO TA approach will allow up to 5% faulty PEs in the accelerator at the expense of less than 1% loss in the AI task’s accuracy.

REFERENCES

[1] V. Sze, Y. Chen, T. Yang and J. S. Emer, “Efficient Processing of Deep Neural Networks: A Tutorial and Survey,” in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017
[2] Y. Chen, T. Yang, J. Emer and V. Sze, “Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices,” in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2019
[3] J. Lee, J. Lee, D. Han, J. Lee, G. Park and H. Yoo, “LNP4U: A 25.3TFLOPS/W Sparse Deep-Neural-Network Learning Processor with Fine-Grained Mixed Precision of FP8-FP16,” IEEE International Solid-State Circuits Conference - (ISSCC), 2019
[4] N. Jouppi et. al., “A domain-specific architecture for deep neural networks,”IEEE Architect. Commun. 61, vol 9, pp. 50â€”59, 2018
[5] AWS Inferenta Chip: https://aws.amazon.com/machine-learning/inferenta/
[6] S. Markidis, et. al., “NVIDIA Tensor Core Programmability, Performance & Precision,” IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2018
[7] G. Ben et. al. “Artificial-intelligence hardware: New opportunities for semiconductor companies,”IEEE Spectrum, April, 2020
[8] L. Tung, “AI-IPU Killer: Google reveals just how powerful its MPU2 chip really is,”IEEE ZDNet, December 14, 2017
[38] R. Ray Ramadorai, et. al., “Method and apparatus for disabling and swapping cores in a multi-core microprocessor,”, Intel Corporation, US Patent: US20070255985A1
[39] online: [http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf](http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf)
[40] Designware IP, online: [https://www.synopsys.com/designware-ip.html](https://www.synopsys.com/designware-ip.html)
[41] Synopsys: [https://www.synopsys.com/](https://www.synopsys.com/)
[42] Opencores: [https://www.opencores.org/](https://www.opencores.org/)
[43] MATLAB: [https://www.mathworks.com/products/matlab.html](https://www.mathworks.com/products/matlab.html)
[44] Pytorch: [https://pytorch.org/](https://pytorch.org/)