TARR: Enhancing the Resilience of Deep Learning Accelerators using Low-cost Tiling-Aware Range Restriction Modules

Mani Sadati

***Abstract*—Deep learning accelerators are increasingly integral to safety-critical applications such as autonomous vehicles and medical diagnosis. However, they are susceptible to transient faults in their systolic arrays, which can cause bit flips, leading to model mispredictions. Ensuring error resilience in such systems is crucial and requires minimal overhead for efficiency.**

In this paper, we introduce TARR (Tiling-Aware Range Restriction), a low-overhead approach to fortify DNNs by integrating range restriction modules within deep learning accelerators. By leveraging the underlying dataflow and hardware characteristics, particularly tiling, we accurately bound inner partial sums in convolution operations. Our method establishes a unique bounding value based on the dataflow, enabling precise detection and mitigation of large faulty values.

Experimental results on widely-used DNN architectures such as VGG16, AlexNet, and SqueezeNet demonstrate the efficacy of our tiling-aware approach. We achieve a balanced trade-off between mitigating accuracy loss and minimizing overhead, making TARR a promising solution for enhancing the resilience of deep learning accelerators in safety-critical applications.

I. Introduction

Deep learning accelerators are increasingly used for deploying deep learning models in safety-critical applications such as autonomous vehicles (AVs) [1]. These accelerators often employ systolic arrays at their core to efficiently handle computationally intensive operations, including matrix multiplications and convolutions.

However, the reliability of these accelerators, especially in safety-critical contexts like AVs, is a significant concern. Systolic arrays are particularly susceptible to hardware faults due to their parallel processing nature [2]. Hardware faults can be permanent (e.g., a stuck-at fault in the data path of a MAC unit) or transient (e.g., bit-flips in the data path). Permanent faults in systolic arrays may arise from physical damage, manufacturing defects, or wear and tear over time. In contrast, transient faults are typically caused by high-energy particles, such as neutrons and alpha particles, striking the hardware. Unlike permanent faults, transient faults are unpredictable in their occurrence and location, necessitating robust defense mechanisms. Even a single erroneous output in applications like AVs can lead to catastrophic outcomes, for instance, Fig. 1 illustrates a case where a red light is falsely labeled as green in an AV due to faults present in its deep learning accelerator.

Previous efforts to enhance the fault tolerance of DNN accelerators have largely focused on techniques such as triple Modular Redundancy [3], algorithm-based fault tolerance, and checksums [4]. While effective, these methods introduce

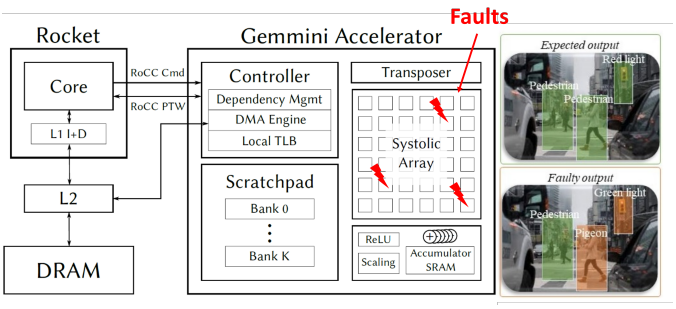

Fig. 1. A red light is falsely labeled as green in an AV due to faults present in its deep learning accelerator.

significant overhead. An alternative approach, range restriction [5], aims to keep calculations within a safe range to prevent error propagation. This method is based on the premise that if an error significantly alters the magnitude of outputs, it is likely to result in a wrong output; if not, the error is likely benign. However, this approach has traditionally been applied either universally across all processing elements, which incurs substantial overhead, or only to the final outputs of operations, neglecting intermediate computational stages [6], [7].

In this project, we introduce Tiling-Aware Range Restriction (TARR) modules that apply range restriction after processing each operation tile in a convolutional layer. TARR leverages two inherent characteristics of DNNs and their accelerators: 1) DNNs' resilience to minor perturbations, and 2) the sequential, accumulative nature of systolic array operations due to operation tiling. Previous implementations of range restriction have either introduced high overhead by covering each processing element or failed to adequately control fault propagation by limiting the restriction to final outputs. By incorporating unique range restrictions after processing each operation tile, TARR utilizes DNNs' natural tolerance for small perturbations and the progressive accumulation in tile outputs—where output values increase from earlier to later tiles. This module allows for early error detection and correction, minimizing inaccurate calculations and enhancing the reliability of DNN accelerators with a cost-effective fault mitigation strategy.

The main contributions of this project are as follows:

- We introduced TARR, a novel tile-wise range restriction mechanism that effectively leverages the underlying hardware structure of systolic arrays to enhance fault resilience in deep learning accelerators.
- Our approach significantly improves fault mitigation,

striking a balance between the granularity of fault detection and the overhead incurred, which has been demonstrated to be the optimal trade-off in our evaluations.
- We proposed an innovative method for applying different granularities of range restriction to various layers within a model, further optimizing performance and efficiency.
- Experimental results indicate that TARR can be integrated into existing deep learning accelerators with negligible overhead, while substantially reducing the risk of safety hazards due to mispredictions caused by hardware faults.

The rest of this paper is organized as follows: Section II provides a background on the foundational concepts pertinent to our work. In Section III, we delineate the fault model that underpins our experimental analysis. Section IV delivers a comprehensive overview of the core methodology employed, alongside the simulation challenges encountered. Section V presents a thorough evaluation of our fault mitigation strategy, examining both the overhead implications and a layer sensitivity analysis to illustrate the efficacy of our approach. Section VI concludes the paper.

II. Background

A. Fault Mitigation

Modern strategies to increase fault tolerance in deep neural network (DNN) accelerators typically employ methods such as redundancy, error correction codes (ECC), and advanced software-based detection and mitigation techniques. Notably, Triple Modular Redundancy (TMR) [5] and ECC [3], [8] are prevalent for safeguarding against transient faults, although at the cost of increased space and energy consumption. Additionally, newer studies have ventured into quantization and selective precision scaling as ways to bolster fault resilience more efficiently [8], though these solutions can necessitate intricate hardware adjustments or compromise on model accuracy. Other notable methods include Algorithm-Based Fault Tolerance (ABFT) [3], [4].

A powerful idea in this area is range restriction which has been explored by [5]–[7], [9], [10]. The idea of range restriction is to utilize the inherent resiliency of DNNs to small perterbuations. The gaol is to correct the large faulty values that lead to silent Data Corruptions (SDCs) or misspprediction to benign faults which do not change the output of the model. As depicted in Fig. 2, if a number in the output of a convolutional layer exceeds a predetermined maximum value, it is deemed out of range and will be clamped to either the maximum value or 0, depending on the data distribution in the layer output. This maximum value is determined during the model's training phase, similar to model weights. Negative numbers do not need to be considered because all numbers less than 0, whether erroneous or not, will be clamped to 0 by the ReLU activation function.

Fig. 3 shows an example of restricting the value of an out-of-range output to 0. Initially, the number has a value of 0.5. However, a transient fault causes a bit-flip, changing its value to 4.5. If the maximum value set is 4, then 4.5 will be recognized as out-of-range and clamped to 0. Note that range restrictions do not perfectly restore the erroneous

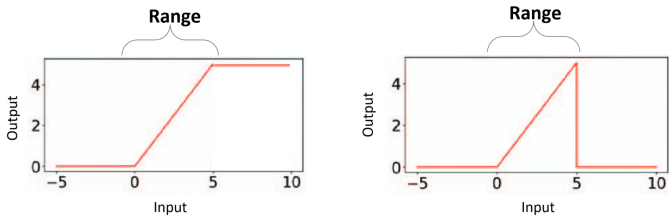

Fig. 2. Range restriction methods that clamp the output to the maximum value or to 0.

number back to its original, error-free value; instead, they reduce the magnitude of the error. This strategy leverages the intrinsic fault tolerance of a DNN: perfect correction is unnecessary as errors with small magnitudes will be benign in a DNN.

Fig. 3. An example illustrating a case where a bit flip in output caused its value to be out of range, and is hence clamped to 0.

So far, this idea has been applied either universally across all PEs which poses a significant overhead or only to the final outputs of the operation, overlooking intermediate computational stages [6], [7].

B. Operation Tiling

Operation tiling in a systolic array refers to the way in which a computation is divided into smaller sub-computations, or tiles, which are processed by the array in a parallel and pipelined manner. Operation tiling is useful when the computation is too large to be performed by a single processor. By dividing the computation into tiles, the systolic array can process the computation in a streaming fashion without having to store the entire computation in memory at once. This can be useful for tasks such as image and video processing, where the computation is often too large to be processed in one processing pass.

III. Fault Model

Figure 1 shows Gemmini's architecture and our fault injection setup. We consider a multiple transient bit-flip fault model to study the effect of transient faults in the MAC units of the systolic array. Transient faults in systolic arrays can be caused by various factors, including voltage fluctuations, temperature variations, radiation, and manufacturing variabilities. More specifically, we set the following constraints in our fault model:

1) We do not consider faults in the memory elements as they can be protected with ECC.

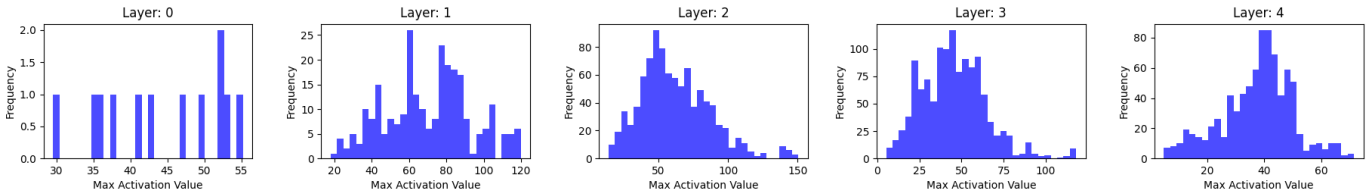

Fig. 4. Histogram of maximum values of partial sums for different tiles in each layer of the AlexNet.

2) We only consider faults in the data path because they can pass silently and lead to incorrect output without detection.
3) We focus on faults in the MAC units as they constitute the majority of the hardware area of the data path in systolic arrays.
4) We consider a bit-flip fault within the process of each output channel tile. This way we will have a balanced fault model where the layers with more computation (and more tiles) will have higher chances of having a fault injection.
5) We consider 32-bit fixed-point representation of values in the systolic array. Although our approach can be applied to other value representations as well.

IV. Methodology

A. Overview

In this section, we delve into the rationale behind adopting tile-aware range restriction and elaborate on the construction and integration of tile-aware range restriction modules within deep learning accelerators.

B. Motivation

Our investigation begins with the hypothesis that computational tiles within a deep learning accelerator yield distinct value distributions during inference. This variance is pivotal: certain tiles may consistently output higher magnitude values, while others typically generate lower magnitude outputs. The cornerstone for setting range restriction parameters is the maximal output value produced during fault-free inference, as range restriction methods use this as an upper bound to ensure legitimate values are not erroneously flagged as faults. Anything exceeding this threshold is likely erroneous and requires correction.

An analysis, illustrated in Figure 4, reveals compelling trends within the AlexNet model's layers. The histogram of maximum output values per tile during full dataset inference presents a stark contrast. For instance, in layer 2, some tiles peaked near zero, whereas others approached 150. Standard layer-wise approaches would default to the higher value (150) for all computations. Our approach harnesses these discrepancies in maximum values, proposing that individual tiles have tailored upper bounds.

C. Fault Correction

Identifying faulty values is only half the battle; determining the appropriate correction method is equally crucial.

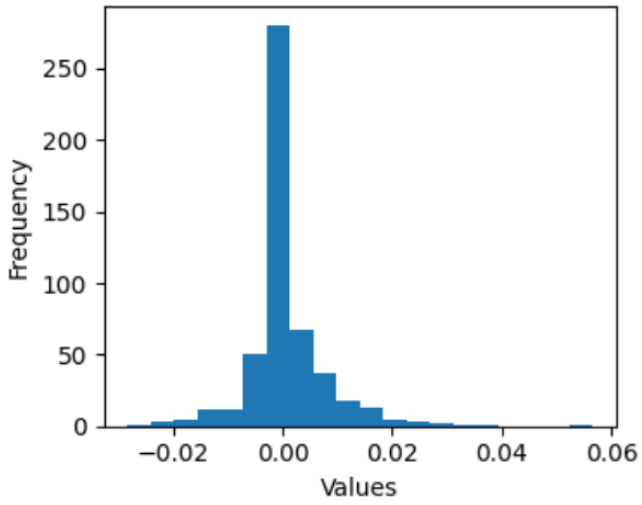

Fig. 5. Histogram of output values in each layer 1 of the AlexNet.

Previous research typically suggests reassigning these to the upper bound value. However, our observations indicate a different strategy may be more effective. Based on Figure. 5,for AlexNet, the preponderance of output values within a layer gravitate towards zero. This consistent low-value trend suggests that zero, statistically, is the most probable correction value, challenging the traditional approach of using the upper bound as a corrective measure. By adopting this tile-aware correction strategy, we aim to more accurately reflect the natural output distribution, thereby enhancing fault tolerance without compromising the integrity of the accelerator's computations.

The final format of the range restriction operation can be seen in Equation 1. N is the number of different tiles indicating different boundaries (λ_i) for each tile.

$$F(x) = \begin{cases} 0 & \text{if } x > \lambda_i \\ x & \text{if } x \leq \lambda_i \end{cases} \quad \forall i \in \{1, ..., N\} \tag{1}$$

D. Hardware Implementation

Implementing the TARR methodology involves integrating it with the hardware, particularly where it can be most effective. After each tile in the systolic array finishes its computations, the results go to an accumulator SRAM. This is where the data is gathered before moving on to the next phase of processing. As shown in Figure 6, we chose this point—the spot where data exits the systolic array and enters the accumulator—to place our TARR modules.

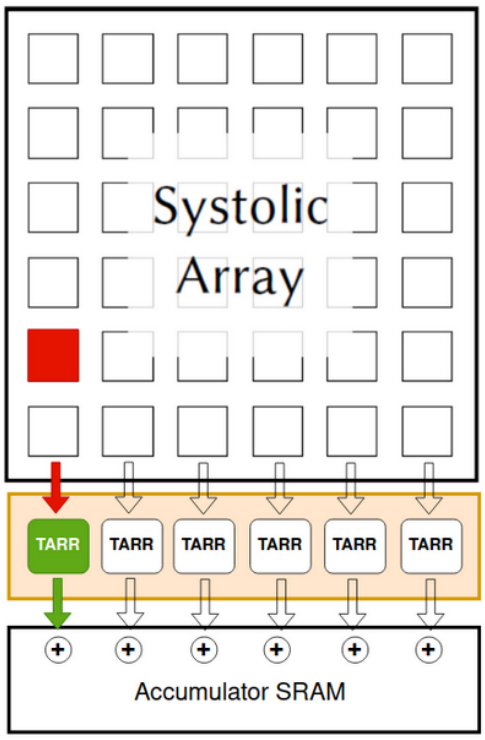

Fig. 6. Hardware implementation of TARR modules.

The TARR module has a comparator that does the job of checking the computed value against an upper limit set for that tile. If the value is too high, the module flags it as a fault and corrects it to zero, which is often the most likely correct value. This setup allows us to catch and fix errors right where the data is transitioning, making sure that faulty values are corrected before they move any further in the process.

V. Simulation Challenges

One of the key challenges that we faced was simulation time of fault injection and our mitigation technique in a RTL-level was significantly time-consuming. For example a simple infernce of a Resnet50 model took several days. Therefore, we face toward simulation of our work in software frameworks that are both fast and popular such as pytorch.

A. Software simulation of tiled convolutional layers

The fundamental operation of a convolution layer involves sliding K different $C \times R \times S$ element filters (where C is the number of input channels, and R and S are the height and width of the filter, respectively) across a $C \times W_{in} \times H_{in}$ element input activation map (where W_{in} and H_{in} are the height and width of the input, respectively) to generate a $K \times W \times H$ element output activation map. Here, $W = W_{in} - R + 1$ (assuming a stride of 1 and no padding) and $H = H_{in} - S + 1$ are the width and height of the output, respectively. By applying multiple filters (K) to the input activation maps, K distinct output channels are produced. This operation can be expressed using nested loops, as shown in Fig. 7.

```
For k = 0 to K
 For c = 0 to C
  For w = 0 to W
   For h = 0 to H
    For r = 0 to R
     For s = 0 to s
        out[k][w][h] +=
            in[c][w+r][h+s] *
            w[k][c][r][s];
```

Fig. 7. Nested loops computing a convolution layer. The green loop corresponds to the iteration of different output channels, and the red loop corresponds to the iteration of different input channels.

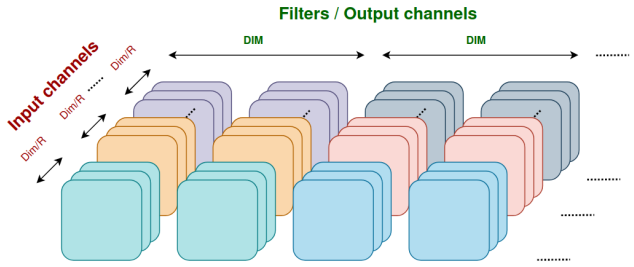

Fig. 8. Tiling scheme for weight stationary convolution. Weight parameters with different colors correspond to different tiles.

```
For oct = 0 to OCT
 For ict = 0 to ICT
  For k = oct*DIM to (oct+1)*DIM
   For c = ict*DIM/R to (ict+1)*DIM/R
    For w = 0 to W
     For h = 0 to H
      For r = 0 to R
       For s = 0 to s
         out[k][w][h] +=
             in[c][w+r][h+s] *
             w[k][c][r][s];
```

Fig. 9. Nested loops computing a convolution layer considering operation tiling. The first green/red loop corresponds to the iteration of processing different tiles with respect to output/input channels, while the second green/red loop corresponds to the iteration of different output/input channels inside a tile. OCT is the number of output channel tiles (K/DIM), and ICT is the number of input channel tiles ($C/(DIM/R)$).

However, this algorithm does not consider operation tiling, which means we do not have access to the target partial sums that are generated right after each operation tile. In order to implement the effect of tiling in our software-based convolution algorithm, we need to observe the actual datapath in the systolic array to see how the tiling divides the operations. We

observed that operation tiling on a weight stationary systolic array divides the weights into several tiles. For example, different filters (output channels) may be processed during different tiles (processing passes). Similarly, different input channels (input feature maps) may be processed in different tiles. Particularly, for our $DIM \times DIM$ systolic array (in our experiments we set $DIM = 16$), we observed that different columns in the systolic array are responsible for different weight filters (output channels), and different rows are responsible for different input channels and filter rows. More specifically, considering the column corresponding to filter k, the first R processing elements correspond to weights in input channel zero (from $w[k][0][0][0]$ to $w[k][0][R][S]$), the next R rows correspond to weights in input channel one (from $w[k][1][0][0]$ to $w[k][1][R][S]$), and so on. Since there are only DIM processing elements inside one column, only a smaller portion of input channels can fit inside one tile (DIM/R). This means that operation tiling divides the convolution weights into two dimensions (as illustrated in Fig. 8): each tile can process only DIM filters, and for each filter, only DIM/R input channels can be processed in one tile[1]. Hence, the convolution nested loops are modified to what is shown in Fig. 9 with the iteration of different output channels broken into two loops (one for different tiles of output channels and one for output channels inside a tile), as well as the iteration of input channels (one for different tiles of input channel and one for input channels inside a tile).

B. TARR module software implementation

Similar to our tiling-aware convolution algorithm shown in Fig. 9, we first profile the maximum partial sum value generated for each $tile(oct, ict)$ (the operation tile corresponding to the oct_{th} subset of output channels and ict_{th} subset of input channels) in a fault-free experiment and store it as $Bound(oct, ict)$. Then, during the fault injection experiment, we check if any partial sum value at the end of $tile(oct, ict)$ exceeds $Bound(oct, ict)$. If so, this indicates a faulty partial sum, which we mitigate by setting the partial sum to zero. Fig. 10 illustrates how we incorporate TARR modules within the convolution nested loops.

```
For oct = 0 to OCT
 For ict = 0 to ICT
  For k = oct*DIM to (oct+1)*DIM
   For c = ict*DIM/R to (ict+1)*DIM/R
    For w = 0 to W
     For h = 0 to H
      For r = 0 to R
       For s = 0 to s
         out[k][w][h] +=
             in[c][w+r][h+s] *
             w[k][c][r][s];
    if out[k][w][h] > Bound(oct,ict)
      Out[k][w][h] = 0
```

Fig. 10. Applying TARR modules to the convolution layer. After the partial sums in each tile are fully calculated and ready to go to the accumulator (before starting the next tile), we apply our tiling-aware upper bounds ($Bound(oct, ict)$).

VI. Evaluation

A. Experimental Setup

ML Applications: We selected three convolutional neural networks—AlexNet, VGG16, and SqueezeNet—for image classification tasks. These models were chosen for their diverse sizes and architectural characteristics and are among the most widely used in vision processing.

Dataset: For this study, we utilized pre-trained, publicly-available models for each architecture to focus on reproducibility and simplify our workflow. The DNNs were pre-trained on the ImageNet dataset, which includes images from 1,000 classes. Table I displays the characteristics of these models.

TABLE I
Characteristics of the studied CNNs.

Benchmark Name	# Conv. Layers	# Parameters	Accuracy (%)
VGG16 [11]	13	527M	69.06
AlexNet [12]	5	58M	54.37
SqueezeNet [13]	26	1.3M	55.62

Metrics: We measure the efficacy and performance overhead of TARR. We chose to report the Top-1 accuracy for image classification as it directly illustrates the efficiency of the fault mitigation technique in converting severe defects (SDC faults) into benign faults.

Overhead: For the performance overhead of TARR, we measured the memory footprint required to store the range restriction values as well as the estimated area overhead for implementing hardware-based TARR modules.

System Setup: Software simulation experiments were conducted on an Intel Xeon E-2224 CPU at 3.40 GHz with 4 cores, with no GPUs. Even without using GPUs we could reach to inference time of around 1s per image batch due to our software optimization techniques.

B. Fault Injection Results

To demonstrate the efficacy of TARR, we assessed model resilience by comparing it with granularity methods proposed in state-of-the-art research. Layer-wise granularity has been explored in [14], [15], and neuron-wise/PE-wise approaches have been discussed in [10]. Table II presents a comparative analysis of accuracy loss and fault detection rates using these methods. An unprotected model exhibits a dramatic reduction in accuracy to nearly 0%, underscoring the importance of fault resilience in safety-critical applications. A trend across different granularities of range-restriction techniques shows that while all methods significantly improve fault mitigation, coarse-grained approaches are typically less effective than

[1]There is also another dimension of tiling where different filter columns (S dimension) are also separated into multiple tiles, but for the sake of having lower overhead and more general implementation, we do not consider them here and stick to only the two tiling dimensions of output channels and input channels.

TABLE II
EFFICIENCY OF RANGE RESTRICTION TECHNIQUES FOR FAULT DETECTION AND MITIGATION.

Benchmark	Fault-Free	UnProtected	Layer-wise Range restriction		**Tiling Aware Range Restriction**		Layer-wise Range restriction	
	Accuracy (%)	Accuracy (%)	Detection (%)	Accuracy (%)	Detection (%)	Accuracy (%)	Detection (%)	Accuracy (%)
VGG16 [11]	69.06	0	5.0	67.81	6.3	69.06	8.5	69.06
AlexNet [12]	54.37	0	5.4	52.50	6.4	54.06	6.9	54.12
SqueezeNet [13]	55.62	0	4.3	50.62	6.6	53.12	7.2	53.28

TABLE III
MEMORY AND AREA OVERHEAD FOR DIFFERENT GRANULARITIES OF RANGE RESTRICTION TECHNIQUES.

Granularity	Storage Overhead (# Parameters)			#Hardware
	VGG16	AlexNet	SqueezeNet	Modules
Layer-wise	13	5	26	N
Tile-wise	20.6K	3.3K	3.5K	N
PE-wise	527M	58M	1.3M	$N \times N$

fine-grained ones. The coarse-grained layer-wise method may truncate an output faulty value to a large positive bound, but this truncated value can still propagate through the network. Similarly, in terms of fault detection, fine-grained range restriction tends to identify more faults, thereby preventing more SDCs and converting them into benign faults.

Moreover, on average, a layer-wise approach achieves a 4.6% detection rate and an accuracy loss of 2.71%. In contrast, our proposed TARR method detects 6.5% of faults—2% more than the layer-wise approach—and reduces the accuracy loss to only 0.9% (1.8% lower). This performance is nearly on par with a PE-wise approach, which has a detection rate of 7.5% and an accuracy loss of 0.8%. TARR's tile-aware range restriction delivers sufficient granularity to match the efficiency of the more fine-grained (PE-wise) approach. The additional 2% of faults detected and mitigated by TARR (compared to the layer-wise approach) were likely to result in SDCs, highlighting the added protection TARR provides. For example, in the VGG16 benchmark, TARR successfully identified and mitigated all faults that would have resulted in SDCs. Although for SqueezeNet, an accuracy loss of 2.5% indicates potential areas for improvement, a PE-wise approach experiences a similar accuracy loss of 2.4%, suggesting marginal benefits from a finer granularity.

An interesting observation from our experiments is that despite over 90% of faults remaining undetected, the majority were benign. This is attributed to the fact that these undetected faults induced minimal value changes within the outputs, to which DNNs exhibit an inherent resilience. This inherent robustness of DNNs to minor perturbations suggests that not all undetected faults are catastrophic, and a certain threshold of fault tolerance is naturally present within these models.

C. Overhead Analysis

Table III presents the memory and area overhead for integrating TARR modules into a deep learning accelerator, alongside a comparative analysis with alternative granularity levels of range restriction.

1) Storage Overhead: Range restriction methods necessitate storing upper bound values; thus, minimizing overhead is crucial to avoid memory transfer bottlenecks and excessive footprint. TARR requires only one upper bound value per operation tile, whereas layer-wise granularity mandates one per layer, and PE-wise/Neuron-wise granularity necessitates a unique upper bound for each processing element within a tile. As indicated in Table III, the layer-wise range restriction introduces negligible overhead, given that most DNNs have fewer than 100 layers, thus significantly reducing storage demands. Conversely, the PE-wise/Neuron-wise approach incurs substantial memory overhead—potentially exceeding the model's parameter size. For example, implementing this granularity on the AlexNet model results in around 58M parameter overhead, equal to the size of its weight parameters, rendering this granularity impractical for real-world applications. TARR, on the other hand, incurs a modest overhead of only 3K-20K extra parameters, a negligible increase when considering the overall system resources.

2) Area Overhead: Comparing area overhead involves assessing the additional hardware modules integrated into the deep learning accelerator for fault mitigation. TARR modules, positioned after each column of the systolic array as comparators, mean that a $N \times N$ systolic array would incorporate only N TARR modules—primarily consisting of comparators. In contrast, layer-wise range restriction methods, typically executed in software, aren't directly comparable in this context as they do not necessitate additional hardware modules. However, values computed at the end of a layer will directly transition to subsequent layers (such as ReLU and Maxpool), avoiding the costly return to DRAM for CPU access. Therefore, a hardware implementation of layer-wise range restriction would require N comparator modules (one for each column) since N output values per cycle will be computed in the systolic array, necessitating parallel comparators equal to the number of columns.

A PE-wise granularity, on the other hand, would incorporate a comparator within each PE, resulting in $N \times N$ comparator modules. For instance, a 16×16 systolic array would necessitate 16 times more range restriction modules compared to TARR. Scaling this to larger systolic arrays, such as 256×256, PE-wise granularity would require over 65,000 modules, significantly outstripping the 256 TARR modules. This contrast underscores the excessive area overhead associated with PE-wise granularity, while TARR and layer-wise range restrictions maintain a much more manageable footprint.

D. Layer Sensitivity Analysis

In this analysis, we explore the efficacy of a tile-wise approach across different layers within a model. Our premise is that not all layers necessitate fine granularity; thus, by

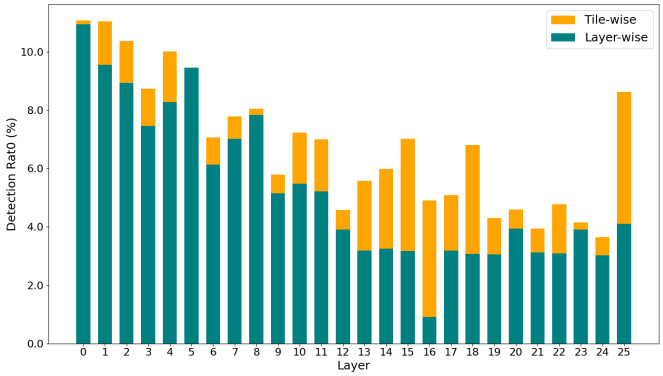

Fig. 11. Detection Rate of TARR and layer-wise range restriction across different layers in SqueezeNet [13].

applying layer-wise bounding selectively, we can potentially reduce overhead further. Figure 11 illustrates the fault detection distribution across the layers of the SqueezeNet model. The data indicates that not every layer benefits from additional granularity—certain layers perform comparably well with layer-wise values. This suggests that these layers are less susceptible to faults, unlike others. Specifically, a significant discrepancy in sensitivity to faults is noted in layers 13 to 18 and layer 25. For instance, the fault detection rate for layer-wise restriction in layer 16 is approximately 1%, in contrast to 5% for tile-wise range restriction.

Based on these insights, we implemented tile-wise range restriction exclusively on the identified sensitive layers, assigning layer-wise upper bounds to the remaining layers. Subsequent experiments revealed that this hybrid approach achieved an accuracy of 54.69%, which is an improvement of over 4% compared to the pure layer-wise method and even a 0.6% improvement from the full tile-wise range restriction. We believe this high improvement comes from our choice of correction value of returning to zero instead of the upper bound, which sometimes might not be beneficial. On the other hand, this optimization reduces the storage overhead from 3.5K parameters to 2.4K, amounting to approximately a 32% reduction in overhead. This strategic selection of granularity per layer further optimizes the balance between computational overhead and fault resilience.

VII. Conclusion

In this work, we investigated the delicate balance between various fault mitigation methods, with a particular focus on range restriction techniques. Our research demonstrates that our tile-aware range restriction (TARR) modules, which capitalize on the specific hardware architecture and dataflow information of the implemented systolic array, occupy an optimal position in this balance, providing both effective fault mitigation and low overhead. Moreover, we have illustrated that adopting different granularities for different layers can further refine this balance, tailoring the approach to the unique needs of each layer. The introduction of our TARR modules showcases the feasibility of enhancing deep learning accelerators with minimal overhead while significantly improving fault tolerance. This work lays the foundation for a new direction in the design of resilient deep learning systems for deep learning accelerators, promising substantial improvements in safety-critical applications.

References

[1] S. Kundu, S. Banerjee, A. Raha, S. Natarajan, and K. Basu, "Toward functional safety of systolic array-based deep learning hardware accelerators," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 3, pp. 485–498, 2021.

[2] Y. Ibrahim, H. Wang, J. Liu, J. Wei, L. Chen, P. Rech, K. Adam, and G. Guo, "Soft errors in dnn accelerators: A comprehensive review," *Microelectronics Reliability*, vol. 115, p. 113969, 2020.

[3] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, "Selective hardening for neural networks in fpgas," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2018.

[4] K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, and Z. Chen, "Ft-cnn: Algorithm-based fault tolerance for convolutional neural networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1677–1689, 2020.

[5] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and 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*, 2017, pp. 1–12.

[6] N.-C. Huang, M.-S. Yang, Y.-C. Chang, and K.-C. Wu, "Decomposable architecture and fault mitigation methodology for deep learning accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2023, pp. 1–8.

[7] M. Taheri, N. Cherezova, M. S. Ansari, M. Jenihhin, A. Mahani, M. Daneshtalab, and J. Raik, "Exploration of activation fault reliability in quantized systolic array-based dnn accelerators," *arXiv preprint arXiv:2401.09509*, 2024.

[8] J. Tan, Q. Wang, K. Yan, X. Wei, and X. Fu, "Saca-fi: A microarchitecture-level fault injection framework for reliability analysis of systolic array based cnn accelerator," *Future Generation Computer Systems*, vol. 147, pp. 251–264, 2023.

[9] M. A. Hanif and M. Shafique, "Faq: Mitigating the impact of faults in the weight memory of dnn accelerators through fault-aware quantization," *arXiv preprint arXiv:2305.12590*, 2023.

[10] B. Ghavami, M. Sadati, Z. Fang, and L. Shannon, "Fitact: Error resilient deep neural networks via fine-grained post-trainable activation functions," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2022, pp. 1239–1244.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. Association for Computing Machinery, 5 2015.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*. Curran Associates, Inc., 2012, pp. 1097–1105.

[13] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and smaller than 0.5mb model size," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. Association for Computing Machinery, 5 2016.

[14] L.-H. Hoang, M. A. Hanif, and M. Shafique, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1241–1246.

[15] Z. Chen, G. Li, and K. Pattabiraman, "A low-cost fault corrector for deep neural networks through range restriction," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2021, pp. 1–13.