Enhancing the Reliability of Split Computing Deep Neural Networks

Enhancing the Reliability of Split Computing Deep Neural Networks

Giuseppe Esposito*, Juan-David Guerrero-Balaguera*, Josie E. Rodriguez Condia*,
Marco Levorato†, Matteo Sonza Reorda*
*Politecnico di Torino - Department of Control and Computer Engineering (DAUIN), Turin, Italy
{giuseppe.esposito, juan.guerrero, josie.rodriguez, matteo.sonzareorda}@polito.it
†University of California - Computer Science Department, Irvine, US
levorato@uci.edu

***Abstract*—[1]Artificial intelligence is becoming increasingly popular for IoT applications in safety-critical fields (e.g., autonomous systems and biomedical, robots). Unfortunately, the inference's workload process alone increases as the model size grows. To meet the computational power limitations of mobile devices running IoT applications, modern services sometimes resort to the Split Computing paradigm. Split Computing divides the inference process of a Neural Network into Head and Tail for their execution in a mobile device and a server, respectively, which also allows the reduction of the overall IoT device's computational cost. Nonetheless, Split Computing can be used in safety-critical fields where reliability is crucial, especially when mobile devices have computational and cost restrictions.**

This paper introduces hardening techniques acting on the software to mitigate the effects of hardware faults on Split Computing models. The proposed hardening techniques consist of *i)* a bounded activation function whose thresholds are refined by training, and *ii)* a per-channel bounding of the bottleneck quantization of the split points. To quantitatively assess their effectiveness, we resorted to two different split configurations of a model for image classification. In addition, we considered a Split Computing model for object detection. Our findings indicate that the proposed approaches effectively reduces fault effects by 3.5% for image classifiers and 5.73% for object detectors when compared with other hardening approaches for general DNNs.

***Index Terms*—Split Computing (SC), Neural Networks, Hardening strategies, Computer Vision, Reliability.**

I. Introduction

Recently, Deep Neural Networks (DNNs) have increased in popularity in the implementation of Artificial Intelligence (AI) algorithms on IoT systems. Due to the high computational power that AI algorithms require to perform inference, their execution often resorts to High-Performance Computing systems or powerful AI accelerators (e.g., Tensor Processing Units, or 'TPUs') operating in parallel matrix multiplications and simpler operations, so significantly speeding up the whole process. Mobile devices typically use Commercial-Off-The-Shelf (COTS) hardware to accomplish the task the DNN is aimed at, which can affect the software implementation and raise concerns for safety-critical applications with strong reliability requirements due to potential hardware faults. AI algorithms are pervasive in a wide range of applications such as avionics [1], nuclear power plants [2], and automotive [3].

Aside from the reliability requirements of IoT applications, the limited capabilities that mobile devices provide often need a careful optimization in terms of memory, power, and energy consumption. The optimization can be accomplished by reducing the complexity of the model using strategies such as quantization, Neural Architecture Search and Knowledge Distillation [4]–[7], facing the challenge of finding the best trade-off between model accuracy and model size. In the literature, workload optimization and model accuracy balancing for mobile devices is achieved by means of Split Computing (SC): a strategy to distribute the DNN data processing between the IoT device and the cloud [8]. Given the total number of layers (L) that compose a DNN, SC consists of deploying the first subset of l layers, denoted as *Head*, on the mobile device and the remaining subset of layers, denoted as *Tail*, on the cloud/edge server [9]. This implies that, during inference, the intermediate output of the DNN (a.k.a. feature map) is sent to a cloud/edge server through a wireless connection. In the case of deep models, the high dimensionality of the intermediate output might not meet the constraints that the bandwidth imposes. For this reason recent studies [10] have proposed tuning techniques acting on the splitting point position between Head and Tail model, along with a compression mechanism (e.g., replacing one layer by one, or a block, of bottleneck layers) that injects an encoder-decoder structure at the splitting point. By doing so the encoder, which becomes part of the Head model, decreases the depth of the output, and the decoder, deployed on the cloud, effectively reconstructs the intermediate output. Although the decreased amount of information that is transmitted through the wireless connection, the depth reduction performed on the encoder's output poses some limitations from the reliability perspective, because of the information loss. Specifically, a possible early corruption of the DNN arising from a hardware fault is likely to impact wider patches of the feature map.

Unfortunately, recent studies have shown that the commercial hardware used by mobile and IoT devices can be affected by hardware faults during in-field operation [11], and the

[1]This work has been supported by the National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing.

979-8-3503-7055-3/24/$31.00 ©2024 IEEE

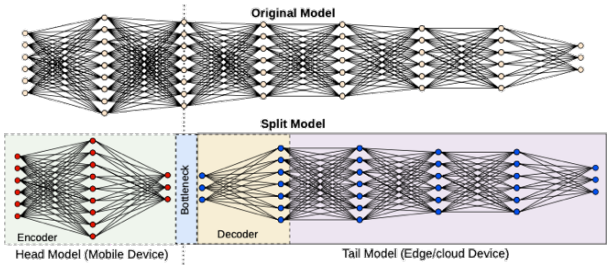

Fig. 1. Supervised Compression for Split Computing design paradigm, [15].

fault probability increases as semiconductors advances. These faults can be caused by internal defects, e.g., due to premature aging, or external impacts like radiation effects. Especially in applications in which the Head model is deployed on mobile devices to support safety-critical applications (e.g., drones and self-driving cars [12]), the sensitivity of the DNN to hardware faults must not be underestimated since they might cause disastrous consequences e.g., car accidents.

Previous work focuses on the reliability evaluation of SC models rather than enhancing their resilience to permanent faults. In [13], the authors proposed an analytical model for estimating the reliability of SC systems. On the other side, authors in [14] experimentally performed a reliability assessment deducing that the average number of predictions affected by faults arising from all the layers deployed on the mobile device can reach 85%. Clearly, these studies do not provide an experimental assessment of their hardening techniques on split computing DNNs or SC-based strategies for reliability improvement.

In this work, we propose, for the first time, two hardening techniques at the application level tailored for SC models (*Adaptive Clipper* and *Saturation Quantizer*). The first technique (*Adaptive Clipper*) enhances the efficacy of a further training step, based on the original SC training pipeline, after restricting the activation function's mapping range. The second solution (*Saturation Quantizer*) explores the effects of a per-channel quantization strategy, where the outliers are cut off from mapped entries distribution. The comparison in SC DNN resilience with two existing techniques adapted to SC architectures showed a noteworthy improvement, which outperforms one of the two reference methodologies.

The paper is structured as follows: Section II presents an overview of SC, supervised compression of DNNs, and strategies used to enforce DNN robustness. Section III describes the proposed hardening techniques along with the fault injection framework for the SC DNN reliability assessment. Section IV details the experimental setup for several split DNN models trained for image classification and object detection tasks. Section V introduces the experimental results. Section VI concludes the paper and discusses future work.

II. Related Work

A. Split Computing

Nowadays, IoT systems have become much more sophisticated with the integration of AI applications. However, these applications usually require more computational power than mobile devices can provide due to their power and computational constraints. An area of recent IoT research focuses on finding a trade-off between the constrained resources of low-power devices and AI model accuracy. Split Computing is a promising architecture paradigm for achieving this goal [12]. Most challenges of SC models are related to identifying the most efficient trade-off among energy consumption, latency, and performance. Given the model (D) composed of L layers, SC divides D into *Head* and *Tail* as indicated in Fig. 1. The Head model consists of the first l layers of the AI model and is executed on the mobile device, generating intermediate outputs. These features are transmitted wirelessly to an edge/cloud server to proceed with Tail's inference. The Tail utilizes the last $L - l$ layers of D to process the input feature map transmitted from the device. The inference result may then be transmitted back to the mobile device to complete the task. The work presented in [15] utilized supervised compression to accomplish the task of lightening the transmitted information through the connection with some modifications to the original neural network architectures. The process entails replacing a segment of the initial architecture of the deep neural network with an encoder-decoder structure at the split point. This results in the encoder becoming a component of the head model on the mobile device, while the decoder becomes a part of the tail model. The feature compression itself is carried out by a series of bottleneck layers in the early stages of the network [10]. One technique for supervised compression is In-network neural compression, also known as Channel Reduction and Bottleneck Quantization (CR+BQ). [16]

One of the most recent trends in object detection architectures is to process intermediate DNN's output to detect elements in the image, but, by doing so, designing any split architecture becomes unfeasible, since the introduction of the encoder-decoder structure would break the forward process of the split model. Object Detectors such as YOLOv4, which present those strong limitations, were not included in the model selection as they would break the entire inference process.

B. Hardening strategies for Split Computing Neural Networks

There exist a variety of techniques to improve the dependability of a system including fault prevention, removal, tolerance, and prediction. Many research works suggest that DNNs have inherent resilience to faults [17] but the safety-critical applications require higher resiliency to hardware faults. Both software and hardware can be designed to mitigate faults that may impact a system's dependability. Therefore, the authors of [18] divided strategies for robustness improvement into four categories: *i)* Model-based approaches: the goal is to derive a model that meets the performance requirements. By construction, when mapped onto hardware, it is capable of tolerating certain hardware-level faults. *ii)* Proactive hardware-based techniques: the objective is to enable the accelerator design to passively tolerate specific hardware-level faults. *iii)* Reactive hardware-based techniques: the objective is to enable the accelerator to respond to faults in real-time, with built-in

monitoring of fault occurrence and low-latency error recovery whenever a fault occurs. *iv)* Cross-layer approaches: the model and hardware share an objective of error tolerance.

The most common approaches for improving the system's reliability fall into the second category. Specifically, one possible idea is to customize activation functions occurring at the end of each convolutional block. Typically, activation functions are clipped with thresholds, based on training dataset statistics (e.g., the absolute maximum detected at each activation function) [19] or according to a fine-tuning algorithm that maximizes a reliability index [20], [21]. Moreover, the activation functions' shape can be customized by smoothing the mapping function at the extreme points with fine-tuned thresholds [22]. The authors of [23] proposed to dynamically update the clipping thresholds to selectively protect layers output based on the corresponding feature maps distributions.

In [24] it is proposed to redesign the maxpool layer of CNNs so as to alter the fault propagation. The authors halt the processing of the frame and move on to the next frame, or use the second largest element if it is reasonably small and if the value of the max element is higher than a threshold. The authors of [25] employed an approach based on clipping all ReLU functions with 6 as the upper bound and then swapping the BatchNormalization layer and, ReLU activation function such that the check is performed directly on the Convolutional layer, which is the most power expensive and then vulnerable to faults.

However, to the best of our knowledge, the effects of the mentioned techniques have never been evaluated on SC DNNs and, consequently, ad-hoc designed hardening strategies have not been proposed, yet. Consequently, this study presents the employed methodology to protect SC Neural Networks trained with CR+BQ supervised compression.

III. Methods

This section proposes two solutions to improve the reliability of mobile devices in SC systems comprising embedded COTS GPUs,: *i)* **Adaptive Clipper**, which performs a further training step that is biased from statistics extracted from the fault-free model, and *ii)* **Saturation Quantizer**, which removes extreme outliers from the input feature map. Moreover, two state-of-the-art hardening techniques, originally intended for general-purpose DNNs, were adapted to the SC models to compare their hardening effects in SC paradigm.

COTS execute the inference of lightweight head models under minimal specialized infrastructures to extend their reliability and fault tolerance. In addition, we assessed the SW-level robustness against hardware-aware faults in the mobile device side as the sole threat to the SC system's reliability. Hence, it is assumed that the communication system and edge/cloud server operate correctly.

As discussed in [26], bit-flips in DNN weights are suitable for modeling the occurrence of a permanent fault in memory elements. However, each neuron of convolutional layers represents the output of an inner product between the input vector $\underline{x}$ and the weights matrix $\underline{w}$. Consequently, flipping the MSB (i.e., bit position with the highest value in a real number binary representation) of a $\underline{w}$'s entry, can cause a substantial change in weight value along with the produced output. The propagation of the mentioned fault leads to a prevalence of critical faults. Therefore, fault injection often leads to values very close to 0 or, in the opposite case, to very high values; this implies that any scalar product involving the convolutions or BatchNormalization operation may result in NaN, ∞, or extremely high values whose propagation is likely to cause system failures. The following sections will first outline the best practices and the main limitations of the SC that are considered during the adaptation of models generally used for the deployment to report a real case. Subsequently, the implementation process of *Adaptive Clipper* and *Saturation Quantizer* is detailed.

A. Split Computing Neural Network models preparation

The depth and, consequently, the not-negligible workload of an inference step deployed from the head of SC models on mobile devices, requires the adaptation of new SC architectures to the SC training framework [27]. The model selection was performed by considering both the suitability of the split in terms of encoder-decoder design and the constrained power resources that the mobile device, used to execute the DNN, is provided. It is currently best practice to use residual blocks in the design of a new feature extractor. This tool utilizes the linear mapping of the input (i.e. skip connections) to the output of the corresponding convolutional block. The redesign of the original architecture must take into account the skip connection, which enables deeper model training [28], [29]. This ensures that the forward pass of the data through the network is not interrupted while the model is still able to learn to accomplish the task it is aimed for.

During the bottleneck design process, the best approach is to not replace the first pretrained convolutional block of the teacher model with the encoder architecture such that the extraction process focuses on target high-level features. Additionally, it is important to keep the bottleneck output feature map spatial dimensions fixed to those of the original layers. This ensures that the Mean Squared Error (MSE) between the corresponding outputs of the subsequent layers can be coherently computed along with the composition of the resulting loss function.

Eventually, the hyperparameter configurations, used during the training phase, are set to closely match the original strategy used for teacher training in terms of learning rate scheduler, data transformations and batch size.

B. Adaptive Clipper

Figure 2 outlines the hardening process of an SC DNN with *Adaptive Clipper*. The Rectified Linear Unit (ReLU) activation function is the particular case of Hard Hyperbolic Tangent (HardTanH) where, given the input feature map $\underline{x}$, the entries are mapped to 0 if $x \leq 0$; otherwise, an identity function is applied. The aim is to replace activation functions at the end

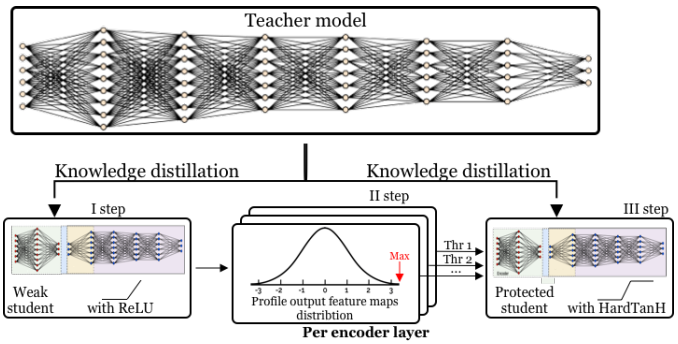

Fig. 2. Adaptive Clipper implementation

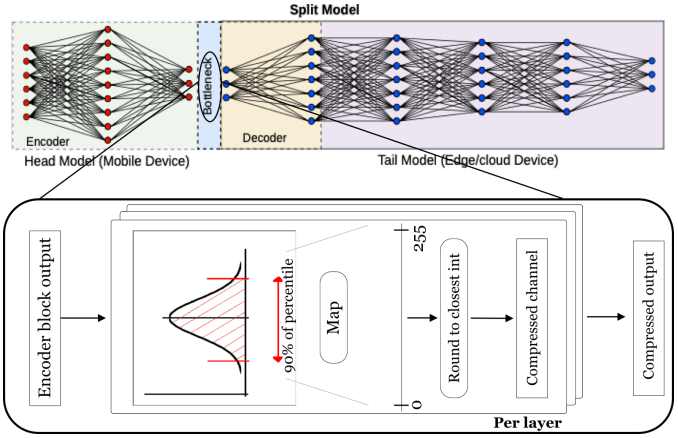

Fig. 3. Saturation Quantizer truncated quantization process.

of convolutional blocks with a clipped version to minimize the resulting overhead.

Given the pretrained student model S, the distribution of the feature maps at the end of each convolutional block is profiled such that their maximum (max) can be computed through an inference step of training data. Consequently, the encoder's ReLU layers are replaced with HardTanH functions where the lower bound is set to 0 and max is used as the upper bound of the HardTanH function according to the step function in Equation 1. This ensures that the model maintains not only the non-negativity property of the ReLU output but also fixes the extreme statistics of our target distribution. Once the new model has been designed, a new training step resorts to the corresponding not-hardened SC DNN training strategy. By doing so, the weights are adjusted at each parameter update according to the new reference distribution statistics improving the inherent robustness of the system.

$$\text{HardTanH}(x, max(\underline{x})) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } 0 < x < max(\underline{x}) \\ max(\underline{x}) & \text{for } x \geq max(\underline{x}) \end{cases} \quad (1)$$

C. Saturation Quantizer

Supervised Compression for Split Computing [15] relies on the channel-reduction and quantization of mid-level feature maps (a.k.a. Neural Compression) to avoid a bottleneck in the wireless connection through the injection of an encoder-decoder architecture. This includes the quantization of the encoder block output by mapping the tensor entries to the interval $[0, 255]$, where the min and max of the input distribution correspond to the lower and upper bounds, respectively. 255 is typically chosen as the upper bound such that when the mapped tensor is cast to int8, the entries distribution cover all the available bit-space while the information loss is minimized. Eventually, mapped data are cast to int8. Such quantization is performed with statistics (max and min) computed at runtime. Then, when a fault has significant effects on mid-level outputs (e.g., sharp increase), the distribution of input tensor entries may be significantly skewed towards small values due to the presence of outliers, which then become the maximum value of the distribution and max subsequently mapped to 255. As illustrated in Fig. 3 the aim of the *Saturation Quantizer* technique is to cut off the outliers from input entries distribution by truncating it before 5^{th} and after the 95^{th} percentiles. By doing so, *Saturation Quantizer* is able to cut extreme outliers off from the distribution and avoid the sharp increase of the skewness degree in the input distribution. The beneficial effects of such a technique are further enhanced by computing the statistics of interest from the distribution designed by each channel of the feature map. This implies an increase in the computational cost that is tolerable since the quantization is executed only once per input batch.

IV. Experimental Setup

The evaluation of our hardening approaches are tested through a set of Fault Injection campaigns on the layers' weights tensor of the SC Head model to observe the impacts of permanent faults on two tasks: *i) SC MobileNet V3 Small Classifier* for image classification with 2 split configurations which involve channel reduction to 6 and 12 channels tested on CIFAR10, and *ii) SC Single Shot Detection (SSD)* 300 with backbone VGG16 for object detection with channel reduction to 6 channels tested on COCO 2017. We specifically used the Channel Reduction + Bottleneck Quantization (CR+BQ) method, which involves the point-wise quantization of the encoder output feature map from floating-point to 8-bit format.

To compare our approaches (*Adaptive Clipper* and *Saturation Quantizer*) with state-of-the-art techniques, we adapted *Ranger* [19] and *Swap ReLU6* [25] (based on the swap Batch-Normalization and ReLU operators within a convolutional block), to SC paradigm and used them as a reference for the resiliency improvement of SC models. Since the results from the original works of the mentioned strategies reported beneficial effects on the image classifier, the hardening effect of such strategies was tested, in the Split Computing paradigm, for the 2 split configurations of the SC MobileNet V3 Small Classifier. Moreover, both the object detector and the image classifier are compared with the non-protected DNN (baseline). The considered metrics of interest are the overhead that the different implementations add to the baseline, the percentage of faults that the protected model can cover over

all injected faults, and the relative degradation of the metric of interest for each task. Specifically, for image classification, we considered the Mean Relative Accuracy Degradation (MRAD), and for object detection we considered the Faulty Intersection over Union (Faulty_IoU), which is the ratio between the overlapping pixels over the total number of pixels between the bounding box predicted by the golden model and the bounding box predicted by the corrupted model.

The object detection Fault Injection campaigns were performed, on a 6 node cluster with 2 Intel 16-cores Xeon Scalable Processors Gold 6130 2.10 GHz, and equipped with 6 NVIDIA Tesla V100 SXM2, and 32 GB of RAM. Similarly, the Fault Injection campaigns related to image classification tasks were deployed on a workstation HP Z2 G5 with an Intel Core i9-10800 CPU with 20 cores, 32 GB of RAM, and equipped with an NVIDIA Ampere GPU RTX 3060TI.

To conduct reliability evaluation on SC DNNs, our fault injection framework resorts to a runtime perturbation tool for PyTorch-implemented models [14] based on PyTorchFI [30]. This framework enables the perturbation of either a DNN learnable parameter or a neuron through the use of a user-defined function. The proposed method configures the FI campaign through a configuration file that includes operational specifications, such as the DNN target model, the layer or set of layers subject to perturbations, and the split point location. This initializes the SC model and sets the target layers for the injection. Moreover, the experiment confidence level can be set, along with the error margin and the fault instances probability, and consequently, the number of faults to inject is computed referring to the analytical model detailed in [31].

Based on the model under testing, the position of the learnable parameters to corrupt is randomly extracted from a Uniform distribution within the weights tensor corresponding to the layer to corrupt. Therefore, the FI campaign is executed, and the stuck-at faults are injected while the model evaluates the test set of the same dataset on which it was trained. The collected inference results are then sent to the report generation module, which performs some data preparation for further analysis. The framework classifies injected faults as *i)* Critical Silent Data Corruption (SDC), when the error induces a misclassification, *ii)* Safe SDC, when the corruption changes the confidence level of the prediction but not the assigned label or *iii)* Masked, when the prediction does not change according to the severity of the effect of the corruption on the model's behavior.

V. Experimental results

This section reports and analyzes the experimental results of Fault injection campaigns on hardened SC models for image classification considering CR+BQ with channel reduction to 6 and 12 channels for SC MobileNetV3Small Classifier where targeted layers (i.e., 0,1,2) are executed on the mobile device with 5% of error margin and 50% of fault instances probability. For object detection, we considered split configuration with channel reduction to 6 channels for SSD300 with VGG16 where targeted layers belong to the Head model (i.e., 0, 1, 2, 3, 4). Moreover, it is provided an estimation of the additional overhead that the adaptation to SC paradigm of the considered techniques implies at inference time. The campaigns performed for both tasks satisfy a 5% error margin and 50% fault instances probability while the confidence level is set to 99% for image classification and to 98% for object detection.

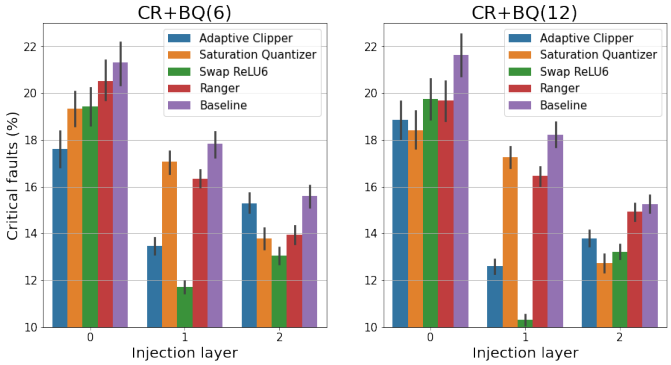

Fig. 4. Percentage of Critical-SDCs on the SC Mobilenet V3 Small Classifier with 2 split configurations: *1)* 6 channels reduction (*left*) and *2)* 12 channels reduction (*right*) per injection layer on the evaluated hardening strategies and the original SC model (Baseline).

A. Image Classification

As observed in the results of Fig. 4 Adaptive Clipper provides a positive effect in the reduction of miss-classifications (from 17.73% and 14.31% less Critical-SDCs) regardless of the split configuration and the injection layer for those models accomplishing Image Classification tasks. Despite SwapReLU6 and Adaptive Clipper are based on the same intuition of Ranger, they overall demonstrate higher effectiveness, which enhances the importance of their additional step of model retraining to adapt the mid-level outputs to the target feature map distributions. Moreover, a slight increase in fault coverage can be noticed in Adaptive Clipper when the split configuration with a reduction to 12 channels is used. In this case, the number of kernels used at the split point and the number of channels of the corresponding output feature map from the teacher are closer, therefore it is easier for the feature extracted from the student to fit the original distribution, limiting the loss of information and consequently increasing resiliency. The beneficial effects of Saturation Quantizer, shown in Fig. 4, are independent from the Split Configuration, showing a constant and moderate trend of fault coverage increase ranging from 1% to 3.2%.

In addition to the coverage, we considered also a suitable metric to compare the performance degradation in the presence of faults. For image classification, we chose the Mean Relative Accuracy Degradation (MRAD) which focuses on the rate of decrease in the Top1 accuracy of the faulty model compared with the Top1 accuracy of the fault-free model (Golden model). Fig. 5 provides key information about the type of faults detected by the different techniques. Specifically, it high-

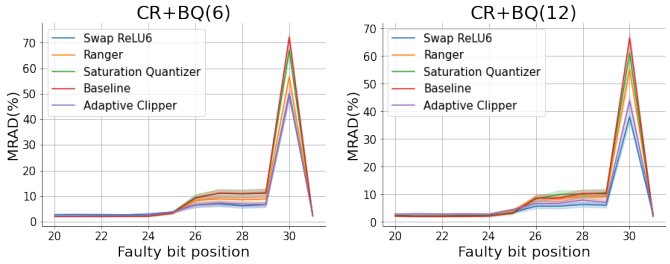

Fig. 5. Mean Relative Accuracy Degradation (MRAD) on SC Mobilenet V3 Small Classifier with 2 split configurations: *1)* 6 channels reduction (*left*) and *2)* 12 channels reduction (*right*) for the evaluated hardening strategies and the original SC model (baseline) when varying the faulty bit position.

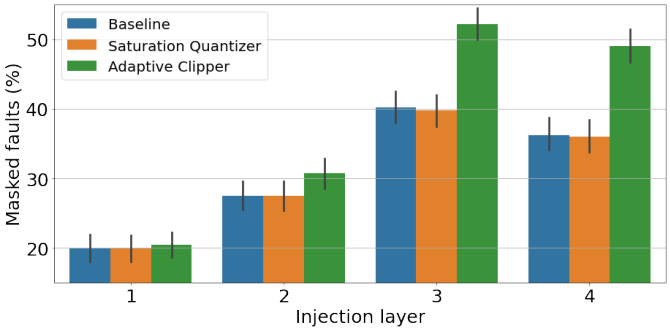

Fig. 6. Percentage of masked faults produced by SC SSD300 with VGG16 object detector with split configuration concerning channel reduction to 6 channels per injection layer by proposed hardening strategy compared to the original SC student model (Baseline).

lights the capability of activation clipping-based techniques to cover the MSB bit-flips as expected from the method presented in section III.

B. Object Detection

Since Object Detectors perform 2 downstream tasks (regression of box edges and box label classification), the severity assessment of the metric degradation focuses mainly on the comparison of the area and on the overlapping w.r.t. the golden model predictions. Then, the evaluation of such models relies on the Faulty_IoU score, the percentage of overlapping over the total number of pixels involved into the golden, and the faulty prediction. Fig. 6 depicts the coverage of Adaptive Clipper along the injection layers reaching 10% of additional Masked prediction w.r.t. the Baseline. On the other side, Saturation Quantizer provides a more limited improvement. The growing beneficial contribution of Adaptive Clipper is due to the higher capability of the fault to propagate through the inference process for the early stages of fault injection. Further analysis that focused on both Critical and Safe SDCs revealed that the consequently decreased number of SDCs has led to the halving of wrongly labeled bounding boxes (from 16.1% to 8.8%) and the decrease of misplaced or misshaped bounding boxes (from 32.0% to 22.1%).

C. Overhead estimate

When the strategy resorts to model-based approaches, there is the risk of seriously increasing the deployed workload on the mobile device since they, in turn, resort to a modification of the original DNN. In fact the hardening techniques might increase the amount of learnable parameters and, consequently, the model size. Table I reports the relative overhead for the proposed strategies. The overhead is computed considering the number of basic operations (addition, subtraction, multiplication, division and comparison) and computing the percentage of increase w.r.t. the baseline model. To make the calculation fairer, an arbitrary weight is assigned to each operation based on: computational complexity, memory access patterns and parallelism and vectorization. It is clear that Adaptive Clipper makes a contribution that is at least comparable to state-of-the-art techniques, despite the Saturation Quantizer adding the lowest overhead to the inference process in both models due to the single execution of bottleneck quantization per inference step. The advantage of Adaptive Clipper lies in the replacement of original activation functions with tuned HardTanH activation functions. This is similar to Swap ReLU6, which also adds the same overhead, while Ranger adds two additional checks after the original activation functions that explain the higher overhead.

TABLE I
ESTIMATION OF OPERATION'S OVERHEAD

SC Model	Ranger	Swap ReLU6	Adaptive Clipper	Saturation Quantizer
Mobilenet V3 Small	0.320%	0.160%	0.160%	0.018%
SSD 300 with VGG16	0.031%	0.015%	0.015%	0.018%

VI. CONCLUSIONS

In this work, we proposed two solutions (*Adaptive Clipper* and *Saturation Quantizer*) to enhance the reliability of DNNs using the Split Computing paradigm. Both solutions focus on protecting the Head computing part (first part of the split), which is typically computed on mobile and embedded devices. The flexibility of the proposed solutions allow the implementation of the *Adaptive Clipper* on all SC-based DNNs, while *Saturation Quantizer* is more effective on all SC models that include mid-level output quantization. In addition, both solutions can be combined to extend system resilience. The experiments evaluated the effectiveness of both solutions as hardening techniques under image classification and object detection tasks. We evaluate both solutions with respect to state-of-the-art strategies considering their fault coverage, metrics degradation and the overall overhead.

Our experiment findings highlight the importance of training when proactive hardware-based techniques (specifically, clipping the activation functions) are used to enhance the inherent robustness of the system masking, on average, 5.73% of hardware faults effects in the object detection model.

As future works, we plan to evaluate the effectiveness of the proposed techniques on hardware-aware fault injection at the neuron level and extend the development of new SC DNN hardening strategies.

REFERENCES

[1] Johnson *et al.*, "A review of fault management techniques used in safety-critical avionic systems," *Progress in Aerospace Sciences*, vol. 32, no. 5, pp. 415–431, 1996.

[2] P. Kumar, L. K. Singh, and C. Kumar, "Performance evaluation of safety-critical systems of nuclear power plant systems," *Nuclear Engineering and Technology*, vol. 52, no. 3, pp. 560–567, 2020.

[3] R. Weissnegger, M. Schuss, C. Kreiner, M. Pistauer, K. Römer, and C. Steger, "Simulation-based verification of automotive safety-critical systems based on east-adl," *Procedia computer science*, vol. 83, pp. 245–252, 2016.

[4] C.-H. Wang, K.-Y. Huang, Y. Yao, J.-C. Chen, H.-H. Shuai, and W.-H. Cheng, "Lightweight deep learning: An overview," *IEEE Consumer Electronics Magazine*, pp. 1–12, 2022.

[5] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[6] D. Das, N. Mellempudi, D. Mudigere, D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas *et al.*, "Mixed precision training of convolutional neural networks using integer operations," *arXiv preprint arXiv:1802.00930*, 2018.

[7] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535–541.

[8] L. Sun, X. Jiang, H. Ren, and Y. Guo, "Edge-cloud computing and artificial intelligence in internet of medical things: architecture, technology and application," *IEEE Access*, vol. 8, pp. 101 079–101 092, 2020.

[9] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[10] A. E. Eshratifar, A. Esmaili, and M. Pedram, "Bottlenet: A deep learning architecture for intelligent mobile cloud computing services," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.

[11] S. Hamdioui, D. Gizopoulos, G. Guido, M. Nicolaidis, A. Grasset, and P. Bonnot, "Reliability challenges of real-time systems in forthcoming technology nodes," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 129–134.

[12] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Comput. Surv.*, vol. 55, no. 5, dec 2022. [Online]. Available: https://doi.org/10.1145/3527155

[13] J.-D. Guerrero-Balaguera, I. A. Harshbarger, J. E. R. Condia, M. Levorato, and M. Sonza Reorda, "Reliability estimation of split dnn models for distributed computing in iot systems," in *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*, 2023, pp. 1–4.

[14] G. Esposito, J.-D. Guerrero-Balaguera, J. E. R. Condia, M. Levorato, and M. Sonza Reorda, "Assessing the reliability of different split computing neural network applications," in *IEEE 25Th Latin American Test Symposium (LATS)*, 2024.

[15] Y. Matsubara *et al.*, "Supervised compression for resource-constrained edge computing systems," in *IEEE/CVF Winter Conf. on Applications of Computer Vision (WACV)*, 2022, pp. 923–933.

[16] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[17] A. Gebregiorgis and M. B. Tahoori, "Testing of neuromorphic circuits: Structural vs functional," in *2019 IEEE International Test Conference (ITC)*. IEEE, 2019, pp. 1–10.

[18] F. Su, C. Liu, and H.-G. Stratigopoulos, "Testability and dependability of ai hardware: Survey, trends, challenges, and perspectives," *IEEE Design & Test*, 2023.

[19] 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.

[20] 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)*. IEEE, 2022, pp. 1239–1244.

[21] J. Zhan, R. Sun, W. Jiang, Y. Jiang, X. Yin, and C. Zhuo, "Improving fault tolerance for reliable dnn using boundary-aware activation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 10, pp. 3414–3425, 2021.

[22] 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)*. IEEE, 2020, pp. 1241–1246.

[23] Z. Liu and X. Yang, "An efficient structure to improve the reliability of deep neural networks on arms," *Microelectronics Reliability*, vol. 136, p. 114729, 2022.

[24] F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on gpus," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2018.

[25] N. Cavagnero, F. Dos Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech, "Transient-fault-aware design and training to enhance dnns reliability with zero-overhead," in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2022, pp. 1–7.

[26] Y. Liu *et al.*, "Fault injection attack on deep neural network," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 131–138.

[27] "Supervised compression for split computing framework," https://github.com/yoshitomo-matsubara/sc2-benchmark.git.

[28] K. He *et al.*, in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[29] A. Howard *et al.*, "Searching for mobilenetv3," in *IEEE/CVF Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 1314–1324.

[30] A. Mahmoud *et al.*, "Pytorchfi: A runtime perturbation tool for dnns," in *50th Annu. IEEE/IFIP Int. Conf. on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2020, pp. 25–31.

[31] R. Leveugle *et al.*, "Statistical fault injection: Quantified error and confidence," in *Design, Automation & Test in Europe Conf. & Exh.*, 2009, pp. 502–506.