Ultra-Low Power and Green TSCH-Based WSNs With Proactive Reduction of Idle Listening

2024 IEEE 42nd VLSI Test Symposium (VTS) | 979-8-3503-6378-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/VTS60656.2024.10538938

Evaluating the Reliability of Supervised Compression for Split Computing

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
{juan.guerrero@, josie.rodriguez@, matteo.sonzareorda@}polito.it
†University of California - Computer Science Department, Irvine, US
levorato@uci.edu

Abstract—[1]**Recent advances in Internet-of-things (IoT) and 5G infrastructures promote new computational paradigms such as *Split Computing* (SC) for deploying Deep Neural Networks (DNNs) on mobile applications. In SC, DNNs are partitioned into *head* and *tail* sub-models that are executed on the mobile device and cloud/edge servers, respectively. Modern SC models resort to *head compression* techniques to balance energy consumption, transmission data, and model size while preserving the outstanding accuracy of large state-of-the-art DNNs. These features make SC DNNs suitable for mobile applications, including safety-critical systems (e.g., self-driving vehicles, autonomous robots, and healthcare equipment), where reliability is a paramount factor mandated by strict safety standards. Despite there are many studies available about the reliability of DNNs, the SC models are still unexplored, especially when hardware faults threaten the operation of a mobile device. In this work, we present for the first time *i)* an application-level fault injection strategy for modeling hardware faults on mobile GPUs executing SC DNNs and *ii)* an evaluation of the resilience of supervised compression methods utilized by SC systems. The preliminary results gathered on some representative benchmark networks and configurations show the feasibility and effectiveness of the approach. They also demonstrate that aggressive compression strategies lead to high accuracy degradation (≈ 40%), increasing the overall vulnerability of the DNN and the system.**

Index Terms—**Internet of Things, Deep Neural Networks, Graphics Processing Units (GPUs), Reliability Evaluation, and Split Computing.**

I. Introduction

In the era of 5G technology and wireless connectivity expansion, distributed computing is increasingly vital for handling resource-intensive deep neural network (DNN) tasks. As infrastructure assistance becomes a central component of even safety-critical applications such as connected vehicles [1], reliable computing strategies turn into an increasingly critical component. In this direction, Split Computing (SC), where DNNs are partitioned into a *Head Model* executed by the mobile device and a *Tail model* executed by the edge server [2]–[4] is emerging as an effective solution to reduce the amount of data transported over wireless channels, thus increasing performance and potentially reducing latency, compared to “pure” edge computing.

Recent contributions in SC have proposed supervised compression of intermediate features to make the *head models* computationally efficient, aiming to balance power consumption, computing load, latency, and channel usage for DNN computations on mobile devices [5], [6]. Nonetheless, to the best of our knowledge, there are no reported works that study the effect of such feature compression strategies on the reliability of the DNN, especially when considering hardware faults that may appear due to a variety of phenomena such as manufacturing process variations, test escaped devices, aging, or harsh environmental conditions [7], [8].

The observation behind the work presented in this paper is that the feature compression used in SC approaches reduces the natural redundancy of DNNs, and so it may negatively impact the resilience of the DNN to such faults arising during the system’s operational life. Indeed, previous works on DNN’s reliability [8], [9] have demonstrated that (transient and permanent) faults affecting hardware structures can induce errors at the algorithmic level of the DNN, resulting in wrong predictions that might lead to system malfunctions, reduced efficiency, or catastrophic events [10].

It is worth noting that only a few works have investigated the resilience of compressed Neural Networks by using quantization or neuron pruning [11], [12]. In fact, several authors’ findings in [11], [13] indicate that quantizing neural networks by using fixed-point representation may positively impact their reliability. Other works [13], [14] have explored quantization and pruning strategies to increase the reliability of DNNs by retraining the model or mapping non-essential neurons to defective processor units, improving in this way the manufacturing yield of hardware accelerators. Nonetheless, new compression techniques (i.e., intermediate features and model sizes) of DNNs for SC scenarios are still fully unexplored in terms of reliability.

On the other hand, the reliability evaluation of DNNs often resorts to fault injection (FI) campaigns at the application level, assuming hardware-agnostic error models (e.g., random stuck-at faults on the weights or bit-flips on the feature maps). Unfortunately, these fault models only describe hardware defects in the system memory [11], leaving unmodeled potential defects in other hardware structures (e.g., computational units). Moreover, they do not allow the differentiation of fault effects among different devices or architectures. Despite the fact there have been some attempts to model fault effects on hardware

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

979-8-3503-6378-4/24/$31.00 ©2024 IEEE

accelerators at the application level [13], [14], none of them considered GPU devices. In fact, Split Computing relies on embedded GPUs to deploy lightweight head DNN models. However, embedded GPUs have a limited amount of computational units, which implies that for a given DNN workload, a single faulty core can potentially corrupt a higher amount of neuron computations compared with high-end computational devices with massive parallelism. It is then fundamental to develop error modeling methods that embrace operative features of GPU devices (e.g., the algorithm mapping into the hardware architecture), to enable the effective resilience estimation of DNNs, including Split Computing models.

In this work, we present the first (to the best of our knowledge) framework to assess the resilience to hardware faults in the context of Split Computing (SC) and supervised compression techniques. In addition, we propose a hardware-aware error modeling methodology that accurately describes the propagation effects from faults on Fused Accumulate Multiply (FMA) cores in embedded GPUs used as mobile devices in SC applications. Our error model strategy corrupts the feature maps of the convolutional and linear layers of DNN architectures, taking into account the GPU device's size in terms of computational cores and the mapping strategy of the algorithm into the GPU device. As a study case to show the effectiveness and practicality of the proposed solution, we considered eight split DNN configurations for ResNet50 (six models using Channel Reduction and Bottleneck Quantization (CR+BQ) and two models using Variational Autoencoders (VAE) with entropy encoding). Our results indicate that both CR+BQ and VAE lead to heavy resilience degradation compared to a non-compressed baseline model, but the VAE models have a better performance against faults than CR+BQ techniques. We also unveil that the bottleneck quantization significantly impacts the reliability of the whole SC system due to hardware faults.

The rest of this paper is organized as follows. Section II provides an overview of SC, supervised compression of DNNs, Graphic processing units, and fault propagation models. Section III describes the proposed fault injection methodology. Section IV describes the experimental setup and presents the results for several split DNN models. Finally, Section V concludes the paper and discusses future works.

II. Background

A. Split Computing

Split computing (SC) distributes the execution of a Neural Network task between a low-powered or constrained device and a compute-capable edge or cloud server. Such a distribution involves splitting the DNN model D consisting of L layers into two sub-models: *head* and *tail*. The head model consists of the first l layers of D, and the tail model takes the last $L-l$ layers of the D. The *head* is executed on the device side, producing intermediate features that are transmitted over a wireless link to an edge/cloud server. The latter executes the *tail*'s inference, taking the feature maps coming from the device as input. Finally, the inference result is possibly sent back to the mobile device to complete the task [2], [3].

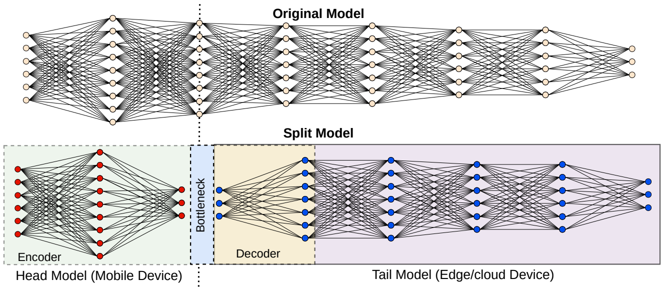

Fig. 1. A scheme of the feature compression for SC using bottleneck layers for channel reduction.

The general goal of SC is to distribute the computing load across multiple devices to optimize the trade-off between energy expense, latency, and task performance. Unfortunately, splitting the DNN without modifying its architecture does not necessarily produce effective operating points. In fact, splitting a network at any of the inner layers often generates high-dimensional features at the output of the head model mapping to a large amount of data to be transmitted [3], [5].

B. Supervised compression of DNNs

Supervised compression methods tackle the problem of reducing the intermediate features to be transmitted between the mobile device and the servers. Such improvement is obtained by inserting encoder-decoder layers at the split point of the DNN, such that the encoder becomes part of the head executed at the mobile device while the decoder is part of the tail model at the edge server.

Fig. 1 depicts a conceptual diagram of intermediate feature compression by inserting bottleneck layers (with a small number of nodes) in the early stages of the network [2]. There exist two primary supervised compression techniques for Split computing based on knowledge destination: *i)* the In-network neural compression (a.k.a Channel Reduction and Bottleneck Quantization (CR+BQ) [6] and *ii)* Variational Autoencoders (VAE) with Entropy Encoding [15]. The CR+BQ encoder-decoder architecture comprises a stacked interconnection of convolution, normalization, and ReLu layer operations. The VAE encoder resorts to an interleave interconnection of convolution and *Generalized Divisive Normalization* - (GND) operations, whereas the decoder uses inverse GND for the feature maps reconstruction.

C. Graphic Processing Units

A Graphic Processing Unit (GPU) comprises a hierarchical structure of computational units called Streaming Multiprocessors (SMs). Each SM integrates several *Scalar/Streaming Processors* (SPs) to perform integer, floating-point, and trigonometric operations. Every SM implements the Single-Instruction Multiple-Thread (SIMT) paradigm, scheduling one instruction for a group of 32 threads (i.e., a *Warp*) per clock cycle. Additionally, an SM includes Local Memories and Register File banks to support the parallel thread execution. Every parallel application (e.g., a CNN) resorts to multiple

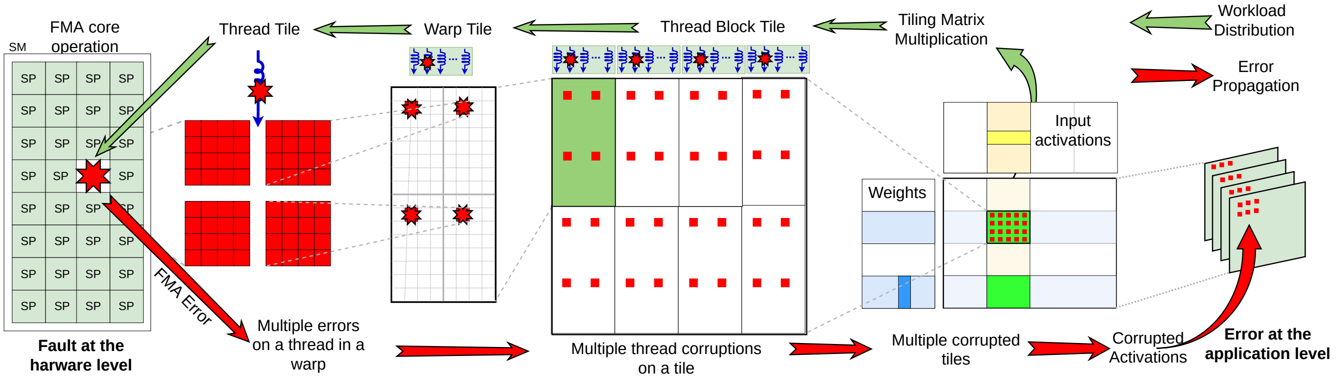

Fig. 2. Illustrative description of error propagation of a faulty GPU's FMA on the tiling Matrix Multiplication

GPU kernels. These kernels consist of several parallel threads organized into "Thread-Blocks" (TBs) groups. When the GPU executes a kernel, the block scheduler submits several TBs in a queue to each SM to optimize the performance. Subsequently, the SM schedules several thread groups (*warps*) into the SPs, and every SP executes the instructions of a thread in a warp, adhering to the SIMT parallel execution model. Recent advances in semiconductor technologies allow to incorporate low-power GPUs in mobile devices, supporting basic machine learning tasks that, in turn, are enhanced by SC techniques [2].

D. Fault propagation and application level errors

In SC scenarios, it is commonly assumed that mobile devices have low-power Commercial Off-the-Shelf (COTS) GPUs [2], [3] that are not developed for safety-critical applications. Hence, the probability that they could be affected by physical defects produced by multiple phenomena (including fabrication imperfections, manufacturing process variation, or aging phenomena [7]) is not negligible. Such defects can produce *faults*, which may occur at any time during the system's operation and are categorized as permanent or transient according to their behavior during the system's operation. Once a *fault* occurs, it can propagate its effect up to the component's outputs, generating *errors* that might be perceived in software as hard or *Silent Data Errors* (SDEs) [16], [17].

These *errors* can propagate through the execution of the application and eventually induce a *failure* of the system (e.g., the system produces a wrong output). In fact, the work in [8] indicates that single permanent faults in datapath cores of GPUs mostly produce single bit-flips at the unit's output at the gate level, and once such fault syndromes are propagated to the instruction level, the DNNs workloads exhibit around 20% of wrong prediction rates when deployed on high-end GPUs. Note that an error may also be masked during propagation through the layers of the full system (i.e., technology, architectural, or software level). Thus, only those fault effects, in the form of errors that reach the software level, can impact the correct execution of the application [18], [19]. In fact, the implementation details of the software layer can determine how a fault-induced error propagates or vanishes during the system's operation. Hence, simulating errors at the software layer provides an acceptable estimation of the impact of hardware faults on the resilience of a DNN.

Error injections at the application level have been widely adopted to study the reliability of DNNs with respect to hardware faults since it is several orders of magnitude faster than detailed low-level hardware fault simulation [11], [18], [20]. Nonetheless, defining accurate error models at the application or system level may result in challenging endeavors without considering the HW/SW interaction. In consequence, in this work, we propose a methodology to describe at the application-level Silent Data Errors (SDEs) produced by different phenomena at the hardware level (e.g., timing defects, wear-out, or permanent defects) affecting the Fused-Multiply-Accumulate (FMA) cores on single SMs of GPU devices. Also, this error model allows the evaluation of SC DNNs when executed in different configurations of mobile GPU devices with different amounts of parallel resources.

III. Fault Injection Methodology

This section describes the methodology and the framework we devised to assess the resilience of supervised compression methods for SC. The proposed approach relies on fault injection (FI) campaigns at the application level, resorting to error perturbations that model the silent effects of the faults in the underlying hardware. It is worth noting that the proposed evaluation method and the simulation framework can be extended to other DNN architectures that use high levels of customization, similar to the ones used for SC.

A. Evaluation flow

As stated before, mobile devices in SC configurations incorporate embedded GPUs, which are dedicated to executing the inference of compact head models. Such devices typically correspond to relatively unreliable COTS components; Hence, they have a non-negligible probability of failing during their operational life. For this work, we assume that hardware faults on the mobile device side are the only threat to the reliability of an SC system (although other threats may exist, for example, connected to software bugs not contemplated in this study); we also assume that the communication system and the edge/cloud server always operate correctly since there might be the possibility to have redundant execution of the *tail* model of the SC system.

In the proposed solution, we rely on FI campaigns by injecting errors at the application level of the head model only, propagating their effects through the full DNN architecture during the inference stage. The evaluation of the injected errors on the DNN is conducted by comparing a golden fault-free inference result with the results obtained from the inference of the perturbed model. This error assessment may depend on the application. In this work, we employed image classification tasks, categorizing the effects of every injected error for every image as *i)* Masked, *ii)* SDC-Safe, and *iii)* SDC-Critical. The first one indicates that the injected error does not change the inference outputs. The second indicates a silent data corruption without prediction changes, and the last one indicates that the injected error induced a misclassification.

B. GPU-aware error modeling for convolution and linear layers

In DNN computations, including SC DNNs, the convolutional and linear layers demand extensive use of dot product operations [21]. The GPUs provide the computational capabilities to support such costly operations by distributing the workload into multiple threads that, in turn, use specialized Multiply-Accumulate (MAC) or Fused Multiply-Accumulate (FMA) cores. However, the continuous utilization of these hardware structures (up to 80% more than other hardware structures [21]) may accelerate their wear out or aging, exacerbating the probability of faults during the system operation.

Furthermore, the intensive use of these computational units also increases the probability that a fault can be activated and propagated, reaching the application in the form of SDEs, for example, due to manufacturing defects, or timing faults. Inspired by the work presented by [8], we propose to model FMA SDEs as single bit-flips corrupting specific regions of the feature maps of convolutional or liner layers. The region of the feature maps to be affected is selected by taking into account the GPU device features (i.e., number of cores, SMs, clusters, etc.), the target algorithm mapped to GPU devices (e.g., Matrix Multiplication), and the algorithm configurations (e.g., degree of parallelism, number of threads, etc).

Typically, convolution and linear DNNs' operations are efficiently implemented in GPUs by resorting to Matrix Multiplication (MM) algorithms. Such implementations divide the MM computation into sub-matrices called *tiles*, which, in turn, are distributed among the GPU's parallel cores. Fig. 2 depicts an exemplification of the tiling workload distribution at different levels in a GPU device. First, the full MM is divided into several *thread block* tile computations, which also are divided into smaller tiles at the warp and thread levels inside the GPU's SMs. This tiling implementation forces every thread in a warp to compute up to four small MM tiles of size 4X4 [22]. Thus, when one SM has a faulty FMA core, the resulting error might propagate to one or more threads per warp, producing multiple data corruptions that are also distributed at the output of the tile. Thus, when more than one tile is executed on the faulty SM, a similar effect will be observed on the results of such tiles.

Considering this error propagation mechanism, we define additional parameters that better describe the error behavior considering different GPU systems. First, we take the tensor output of a target convolutional layer and create a virtual representation of a tiling MM by using three main parameters: the tail size (TS), the block error rate (BER), and the neuron error rate (NER). TS corresponds to the size of every *Thread Block* tile in the layer. BER corresponds to the portion of tiles executed in a faulty SM (eq. 1), and NER corresponds to the fraction of neurons within *Thread Block* corrupted due to one faulty FMA (eq. 2). Finally, specific locations of the feature maps are corrupted by injecting random single bit-flips according to the coordinates defined by the selected tiles assigned to a given SM, which are obtained from the virtual tiling representation.

$$BER = \frac{SMs/GPU}{TotBlockTiles} \tag{1}$$

$$NER = \frac{DotProd/Thread \times Threads/BlockTile}{FMA_{cores}/SM \times WarpTiles \times WarpTileSize} \tag{2}$$

C. Fault injection framework

Based on the approach we described earlier, we developed an in-house framework that wraps highly customized DNNs for SC and automatically orchestrates the error simulation campaigns by creating, organizing, and storing simulation reports with detailed information about every injected error. The simulation environment extends the capabilities of PyTorchFI by adding the hardware-aware perturbation models we listed in the previous section. The FI environment comprises four main modules: *i)* launcher, *ii)* simulation manager, *iii)* injector controller, and *iv)* reports generator. The first module configures and initiates a FI campaign using a configuration file that contains the operational specifications, such as the DNN target model, the layer or set of layers subject to perturbations, the error model parameters (e.g., tile sizes, number of GPU cores, number of warp tiles, etc), and the split point location for the SC models. The *simulation manager* orchestrates the FI campaigns over all the errors considered for the evaluation. The *injection controller* provides an interface between the *simulation manager* and the customized PyTorchFI tool. This component handles the status of the FI and dispatches the inference results to the *reports generation* module. For the sake of reproducibility, our framework is publicly available [2].

IV. Experimental Results

This section presents the experimental results stemming from the adoption of the proposed framework on a sample test case. In particular, we performed a set of FI campaigns at the application level, considering several split configurations based on the ResNet50 architecture for image classification tasks. We evaluated six SC configurations using the Channel Reduction + Bottleneck Quantization (CR+BQ) and two configurations using VAE with entropy encoding. The CR+BQ configurations use channel reduction on the bottleneck to 1, 2, 3, 6, 9, and 12 channels, and all apply a point-wise 8-bit quantization function to the floating-point bottleneck feature outputs. The

[2] https://github.com/divadnauj-GB/SC_Fault_injections.git

TABLE I

Mean Accuracy Degradation results for the fault simulation campaigns with different split configurations

FI Campaign	MRAD (%)								
	BaseLine	CR+BQ						VAE	
		CR(1)	CR(2)	CR(3)	CR(6)	CR(9)	CR(12)	β=0.08	β=5.12
Weights	4.99	8.53	8.29	8.42	8.28	8.36	8.26	8.28	8.61
FMaps	16.10	46.92	49.84	48.18	48.30	49.26	48.02	43.58	39.13

VAE models correspond to two different compression rates defined as $\beta = 0.08$ and $\beta = 5.12$ [15]. We used pre-trained split DNN models taken from a publicly available repository[3]. The DNN models were trained on the ImageNet dataset. For the evaluations, every perturbation was applied on every DNN model while performing the inference of 5,000 images, which corresponds to 5 samples per class of the ImageNet dataset.

The FI experiments were performed acting on the DNN parameters and the output feature maps of the convolutional layers of the head model by injecting errors, as described in the previous section. The parameters' FIs follow the standard statistical approach using a confidence level of 98%, and an error margin of 5% [23]. The injection of errors on the feature maps used the following parameter configurations: tile size (TZ) of 32X32, Neuron Error Rates (NER) per tile of 2%, 4%, 6%, 8%, and 10%. For the experiments, we arbitrarily selected five different Block Error Rate (BER) configurations (i.e., 20%, 40%, 60%, 80%, and 100%), where 100% BER describes an embedded GPU device with only one SM (e.g., Jetson Nano GPU), implying that all tiles of the convolution are susceptible to errors due to a faulty FMA, while 20% BER represents more powerful embedded devices including more SMs (e.g., Jetson AGX Orin with 16 SMs).

For every configuration, we run 65 trials, each time randomly selecting the bit-flip position to be modified. All the experiments were conducted on a workstation HP Z2 G5 with an Intel Core i9-10800 CPU with 20 cores, 32 GB of RAM, and equipped with an RTX 3060TI GPU platform including an NVIDIA Ampere architecture with compute capability (CC) 8.6. The FI campaigns lasted around 1,200 hours for all split configurations.

We used the Mean Relative Accuracy Degradation (MRAD) to evaluate the impact of the injected errors on the performance of every split configuration. MRAD measures the degree of accuracy degradation induced by the injected faults compared to the fault-free scenarios [24]. In addition, we compared the resilience to faults of the split DNN modes with an FI campaign performed on the first four layers of the original version of the ResNet50 model as a baseline.

Table. I reports the MRAD for all SC configurations for every FI campaign targeting the head model of the network. The results show that split models using both CR+BQ and VAE are highly susceptible to hardware faults affecting the weights of the DNN model. In fact, the average accuracy degradation exceeds 8% for all the evaluated configurations, whereas the same evaluation on the baseline model without any compression reports less than 5% of MRAD. The table shows that regardless of the compression technique, the hardware faults inducing data corruption on the parameters induce similar performance degradation ($< 1\%$) on the accuracy of all split models. On the other hand, when considering our SDEs model at the feature maps, the compressed split models show a high MRAD degradation regardless of the compression approach ($> 38\%$), which is quite significant compared to the original model that exhibits a maximum MRAD of 16.1% for the same experiments. Interestingly, the SC models using VAE seem to be more resilient to SDEs than the CR+BQ approaches by around 9%.

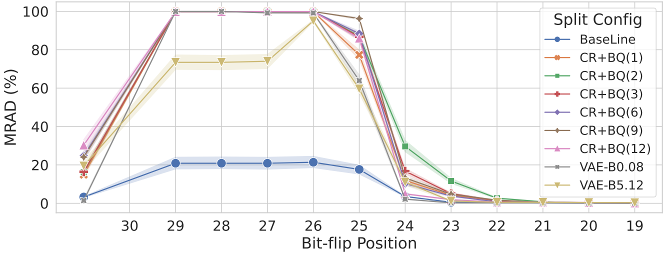

Fig. 3. MRAD for all DNN configurations vs. the bit-flipped position on the feature maps outputs

A. *Evaluation of SDEs in the Feature Maps*

Fig. 3 reports the analysis, at the bit level, of the effect of the proposed GPU-aware SDE modeling applied to convolutional layers of the head part of an SC DNN, assuming the usage of an embedded GPU for computations. It must be noted that bit 30 was removed from the analysis since it always induced 100% of MRAD for all the evaluated models, including the baseline. At first glance, we observe that SDEs corrupting the exponent bits of neuron computations heavily reduce the accuracy performance for all the SC configurations. In fact, when corrupting the bits 25th to 29th, all the compressed models exhibit more than 90% of MRAD, except for VAE-B5.12, which presented around 70% of MRAD. On the other hand, the baseline model presents more resilience to the same SDE corruptions of the exponent (i.e., up to 20% MRAD).

Interestingly, SDEs affecting the two least significant bits of the exponents (i.e., bits 23 and 24) show a moderated but not negligible MRAD that varies from configuration to configuration between 5% (VAE-B0.08) to 35% (CR+BQ(2)). In contrast, the baseline model seems to be resistant to such a kind of errors. In addition, the corruption of the mantissa's bits (i.e., bits 20 down to 0) slightly impacts the accuracy of the SC and baseline models ($< 5\%$ MRAD). Further analysis in terms of the BER is presented in Fig. 4. The obtained results show that for an embedded GPU with 20% of the corrupted tiles (i.e., a GPU with few SMs), all the split models have less than 40% MRAD. Moreover, results suggest that in embedded GPUs with a single SM (BER 100%), the MRAD increases by $\approx 5\%$. Interestingly, the split configurations using VAE compression strategies show more resilience than their CR+BQ counterparts. In general, all the split DNN configurations increase their vulnerability to hardware-induced error as long as the BER increases. However, the VAE $\beta = 5.12$ increased the MRAD by 2% more than the other evaluated configurations when the BER reached 100%.

[3] https://github.com/yoshitomo-matsubara/sc2-benchmark.git

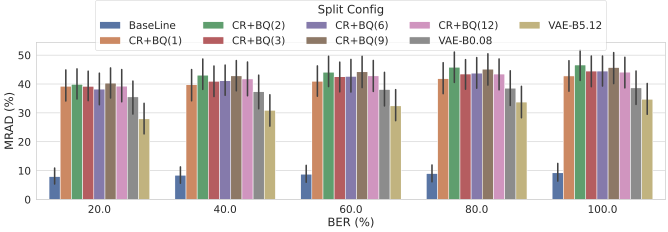

Fig. 4. MRAD for all configurations under different block error rates (BER).

Further analysis indicated that the quantization schemes for the bottlenecks play a crucial role in the reliability of SC models, significantly reducing the resilience of the model to SDE. In fact, the SC with CR+BQ compression uses a simple quantizer to transform floating point representations to 8-bit integers. This quantization heavily distorts the data transmitted to the tail when the magnitude of the corrupted data increases significantly due to SDE affecting the exponent of the upstream layers. Similarly, effects are observed for the SC using VAE compression strategies; still, the stochastic behavior of the VAE architecture benefits the resilience of the model when SDE induces a magnitude of error lower than 2. These preliminary observations indicate that possible mitigation strategies should target the quantization functions by either adopting alternative quantization techniques or including fault-aware saturation mechanisms.

V. Conclusions and Future Work

In this work, we presented a GPU-aware fault model and a framework to assess the resilience to hardware faults of supervised compression techniques used in DNNs for Split Computing (SC) contexts. To the best of our knowledge, this is the first study considering reliability issues in SC systems and providing a solution that allows the designer to take it into account when selecting the best system configuration. Moreover, the error model enables the assessment of the reliability of any SC with different GPU configurations or algorithm implementations. Using the proposed approach, we studied two supervised compression techniques for SC: Channel Reduction + Bottleneck Quantization (CR+BQ) and Variational Autoencoder with Entropy Quantization. The evaluation resorts to application-level FI campaigns to study Silent Data Errors induced by embedded GPU devices while executing the head part of a split DNN. The results indicate that SC DNNs exhibit less resilience to hardware faults than a non-compressed baseline model. The results of the evaluated compression techniques indicate that the VAE architectures show more resiliency to hardware faults than the CR+BQ counterparts. Finally, we observed that the quantization algorithms used by the compression models amplify the distortion induced by hardware faults inducing SDE on the head layers, causing such a high resilience degradation.

In future works, we plan to enhance the bottleneck quantization function of the CR+BQ by adding saturation or intelligent quantization interval detection in order to improve the fault resilience of the overall system.

References

[1] G. Abdelkader *et al.*, “Connected vehicles: Technology review, state of the art, challenges and opportunities,” *Sensors*, vol. 21, no. 22, 2021.

[2] A. E. Eshratifar *et al.*, “Bottlenet: A deep learning architecture for intelligent mobile cloud computing services,” in *IEEE/ACM Int. Symp. on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.

[3] Y. Matsubara *et al.*, “Split computing and early exiting for deep learning applications: Survey and research challenges,” *ACM Comput. Surv.*, mar 2022.

[4] J. Karjee *et al.*, “Split computing: Dnn inference partition with load balancing in iot-edge platform for beyond 5g,” *Measurement: Sensors*, vol. 23, p. 100409, 2022.

[5] S. Singh *et al.*, “End-to-end learning of compressible features,” in *IEEE Int. Conf. on Image Processing (ICIP)*, 2020, pp. 3349–3353.

[6] Y. Matsubara and M. Levorato, “Neural compression and filtering for edge-assisted real-time object detection in challenged networks,” in *25th Int. Conf. on Pattern Recognition (ICPR)*, 2021, pp. 2272–2279.

[7] G. Tshagharyan *et al.*, “Modeling and testing of aging faults in finfet memories for automotive applications,” in *IEEE Int. Test Conf. (TC)*, 2018.

[8] J. E. Rodriguez Condia *et al.*, “A multi-level approach to evaluate the impact of gpu permanent faults on cnn’s reliability,” in *IEEE Int. Test Conf. (TC)*, 2022.

[9] F. F. d. Santos *et al.*, “Analyzing and increasing the reliability of convolutional neural networks on gpus,” *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2019.

[10] R. Possamai Bastos *et al.*, “Assessment of tiny machine-learning computing systems under neutron-induced radiation effects,” *IEEE Trans. Nucl. Sci.*, vol. 69, no. 7, pp. 1683–1690, 2022.

[11] A. Ruospo *et al.*, “Investigating data representation for efficient and reliable convolutional neural networks,” *Microprocessors and Microsystems*, vol. 86, p. 104318, 2021.

[12] M. Sadi and U. Guin, “Test and yield loss reduction of ai and deep learning accelerators,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 1, pp. 104–115, 2022.

[13] “Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping,” *Philosophical Transactions of the Royal Society*, vol. 378, no. 2164, p. 20190164, 2020.

[14] E. Ozen and A. Orailoglu, “Architecting decentralization and customizability in dnn accelerators for hardware defect adaptation,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 3934–3945, 2022.

[15] Y. Matsubara *et al.*, “Sc2 benchmark: Supervised compression for split computing,” 2023.

[16] A. Avizienis *et al.*, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, 2004.

[17] A. D. Singh, “Silent error corruption: The new reliability and test challenge,” in *IEEE Latin American Test Symp. (LATS)*, 2023, pp. 1–2.

[18] 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)*, 2020, pp. 25–31.

[19] M. Kooli *et al.*, “Cache- and register-aware system reliability evaluation based on data lifetime analysis,” in *IEEE VLSI Test Symposium (VTS)*, 2016.

[20] G. Li *et al.*, “Understanding error propagation in deep learning neural network (dnn) accelerators and applications,” in *Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2017.

[21] J.-D. Guerrero-Balaguera *et al.*, “Effective fault simulation of gpu’s permanent faults for reliability estimation of cnns,” in *IEEE Int. Symp. on On-Line Testing and Robust System Design (IOLTS)*, 2022, pp. 1–6.

[22] J. Huang, C. D. Yu, and R. A. van de Geijn, “Implementing strassen’s algorithm with cutlass on nvidia volta gpus,” *arXiv preprint arXiv:1808.07984*, 2018.

[23] A. Ruospo *et al.*, “Assessing convolutional neural networks reliability through statistical fault injections,” in *Design, Automation & Test in Europe Conf. & Exh. (DATE)*, 2023.

[24] S. Hong *et al.*, “Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks,” in *USENIX Conference on Security Symposium (SEC’19)*, 2019, p. 497–514.