On the resilience of INT8 Quantized Neural Networks on Low-Power RISC-V Devices

On the resilience of INT8 Quantized Neural Networks on Low-Power RISC-V Devices

Antonio Porsia
Politecnico di Torino
DAUIN
Turin, Italy

Giacomo Perlo
Politecnico di Torino
DAUIN
Turin, Italy

Annachiara Ruospo
Politecnico di Torino
DAUIN
Turin, Italy

Ernesto Sanchez
Politecnico di Torino
DAUIN
Turin, Italy

***Abstract*—Quantized Neural Networks (QNNs) are increasingly employed to bring Machine Learning (ML) capabilities to edge devices by reducing memory and computational requirements through low-precision arithmetic. QNNs are generally considered more robust to hardware faults, but prior reliability studies, with rare exceptions, have largely relied on high-level fault injection (FI) methodologies that abstract away from hardware details. This paper presents a hardware-level fault injection methodology that integrates Verilator-generated gate-level simulations into QNN inference pipelines, enabling the injection of permanent faults directly into gate-level descriptions of functional units during inferences. Optimizations for small quantization bit widths are also presented to address performance limitations, for instance, leveraging lookup tables and GPU parallelization to speed up simulations. FI campaigns were conducted on multiple networks (MobileNet, MobileNetV2, ResNet18, ResNet34) trained on CIFAR-10 and GTSRB and quantized to INT8 (CIFAR-10 and GTSRB) and UINT8 (CIFAR-10), revealing severe accuracy degradation—averaging a -72.7% drop in classification accuracy - under single stuck faults in an integer multiplier. The results obtained confirm that QNNs may not be as robust as previously thought.**

I. Introduction

Quantization is among the most commonly used techniques to reduce the memory footprint and computational requirements of Deep Neural Networks (DNNs), with the aim of providing advanced Machine Learning (ML) capabilities to edge devices while preserving the network's accuracy. In Quantized Neural Networks (QNNs), weights and activations are represented with lower precision formats, converting 32-bit floating-point numbers to smaller fixed-point numbers, 8-bit integers or even 1-bit values [1]. The efficiency benefits of QNNs are well-known, and their robustness in presence of hardware faults is held to be higher than their floating-point counterparts [2]–[4]. However, reliability assessments of QNNs primarily rely on application-level or software-level fault injection methodologies that abstract away from the complexities of hardware failures. This paper presents a hardware-level fault injection methodology to evaluate the resilience of QNNs against permanent stuck-at faults. Verilator is used to efficiently integrate gate-level simulations inside a ML framework, allowing to inject stuck-at faults into arbitrary gate-level circuits and evaluate their impact on the performance of QNNs without the long times usually needed for gate-level simulations. The proposed approach enables a more accurate assessment of how permanent hardware faults affect the classification accuracy of QNNs, by simplifying the integration of gate-level fault injections into the quantized inference routines of ML frameworks. Section II gives an overall view of QNNs, Verilator and fault assessment methodologies, as well as an overview of reliability studies on QNNs. Section III presents the proposed gate-level fault injection method based on Verilator. Section IV uses the proposed method to inject permanent stuck-at faults into the integer multiplier of a RISC-V core executing several QNNs. Section V provides final remarks and future directions.

This publication is part of the project PNRR-NGEU which has received funding from the MUR – DM 118/2023

II. Background

A. Quantized Neural Networks

Among the techniques devoted to reducing the space and computational requirements of a neural network, quantization is one of the most popular ones, with implementations available in the most popular frameworks, such as Tensorflow and PyTorch. Model quantization involves reducing weights and activations from 32-bit floating-point (FP32) numbers to a lower precision format, such as 16-bit integers (INT16), 8-bit unsigned integers (UINT8) or 8-bit signed integers (INT8) [5], fixed-point numbers (FxP) or even binary values [1] in the most extreme cases. In order to convert FP32 values to their quantized representation, a *quantization scheme* is needed. In general, a quantization scheme takes a floating-point value x and generates an approximate representation by expressing x as a floating-point *scale factor* multiplied by a lower precision value (an integer, in this case) [5]. For the purposes of this work, the *uniform affine quantization* scheme will be used. This scheme defines three parameters: (i) the scale factor s, (ii) a zero point z and (iii) the bit width b. The mapping of a FP32 value x to its quantized representation x_{int} (*quantization*) is computed as follows:

$$x_{\text{int}} = \left\lfloor \frac{x}{s} \right\rceil + z$$

where $\lfloor \cdot \rceil$ is the round-to-nearest function. x_{int} is then clamped between the minimum and maximum value allowed by the bit width b. The *dequantization* step reconstructs the approximate value of x starting from x_{int}:

$$\tilde{x} = s(x_{\text{int}} - z) \approx x$$

Quantization schemes may be applied during training (Quantization-Aware Training, QAT) or after training (Post-Training Quantization, PTQ). In PTQ, the target of this work, a pre-trained FP32 model is quantized to a lower precision format without manipulating the original training process.

B. Verilator

Verilator [6] is a widely used open-source tool that translates Verilog/SystemVerilog listings into C++/SystemC code, enabling fast cycle-accurate simulation of circuits. The resulting "verilated" circuit can then be integrated into existing code, allowing to directly use the results of the simulation in other programs. The latest versions permit dynamically forcing a signal to assume a certain value during the simulation, allowing one to simulate, for instance, stuck-at-0/1 faults.

C. Related work on fault tolerance of QNNs

Fault injection methodologies for DNNs: Reliability assessment of DNNs is usually conducted through fault injection (FI) campaigns, targeting either the DNN application independently of the hardware it is deployed on (*application-level* FI) or the entire system comprising the DNN model and the hardware [7]. Software-level simulation-based FI is one of the most common FI methods due to its low cost, speed, and ease of implementation. In general, simulation-based FI campaigns consist of injecting faults into a DNN model independently of the physical device on which the DNN will be deployed. These faults can be injected either into the DNN model itself without considering the target hardware architecture (*software-level*), or in a DNN model augmented with a description of the target hardware, such as its RTL- or gate-level description (*hardware-level*). The injected faults can be classified as either permanent or transient, depending on whether they persist over time or not. These faults are typically modeled respectively as *stuck-at faults*, representing elements permanently stuck at a certain state, or *bit flips*, representing elements whose original state has been changed.

Fault Injection studies in QNNs: The resilience of Quantized Neural Networks (QNNs) against hardware faults has been extensively analyzed in literature. QNNs are commonly believed to be much more resilient compared to their floating-point counterparts [2]–[4], depending on the level of quantization. However, Gambardella et al. [8] show that a single stuck-at fault in networks quantized at extremely low bit widths (1 or 2 bits) results in accuracy drops of up to 10%, if the network contains convolutional layers, suggesting that convolutional QNNs are actually less robust than previously thought. Neggaz et al. [3] inject bit flips into the quantized weights of AlexNet, VGG16, Googlenet and Squeezenet to simulate memory faults. The study finds that the lower memory and computation overheads of QNNs result in a lower impact of faults on accuracy. Taheri et al. [9] inject single bit flips in the activations of a systolic-array-based simulation of quantized versions of LeNet5 and AlexNet, finding that the drop in reliability of the examined QNNs ranges from negligible (8-bit width) to around 10% (4-bit width). Beyer et al. [10] inject multiple stuck-at faults with various error rates into the registers of the Processing Elements of a Neural Network accelerator, targeting ResNet18 and VGG16 trained on CIFAR-10 and quantized at a bit width of 8 bits. The authors find that at an error rate of 2.1%, the median accuracy drops by 10% and completely degrades at a rate of 5.5%.

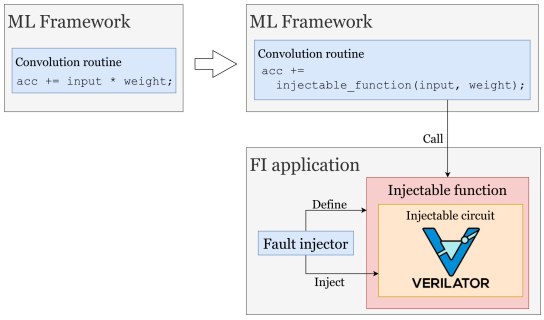


Fig. 1: Example of fault injection in multiplications inside a C++ convolution routine

III. Proposed Method

A. Fault Injection methodology

The proposed FI methodology leverages Verilator to instantiate the gate-level model of a targeted functional unit into inference routines, such as convolutions. In this manner, the operations that would be performed by the functional unit on a physical device are substituted with a gate-level simulation. In general, the proposed method is made up of three phases:

1) **Verilation**: The target functional unit is synthetized to obtain its gate-level description. Before running Verilator, all `wires` in the Verilog gate-level description are marked as `forceable`. The design is then verilated to obtain a C++ model of the circuit, which is then wrapped into a C++ class (the *injectable circuit*) that allows one to inject permanent stuck-at faults into the underlying circuit by specifying wire and value.
2) **Integration**: The source code of the routines of interest in the target ML framework is analyzed to find possible *fault sites*, lines of code containing an operation that on a physical device would be executed by the targeted functional unit. All fault sites are substituted with a call to a user-defined function (the *injectable function*) that uses the injectable circuit to perform the same operation.
3) **Injection**: Once the Verilator-enhanced ML framework is in place, FI routines can be developed independently. The user can easily inject and clear faults in the injectable circuit, as well as redefine the injectable function between inferences.

An example is given in Figure 1, where multiplications are substituted by calls to a Verilator integer multiplier model.

It should be noted that this technique is not limited to integer numbers: the same procedure can be used with different

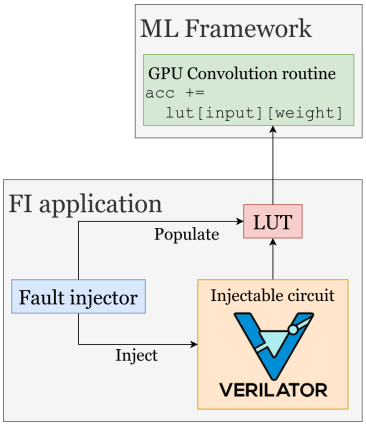


Fig. 2: Example of LUT-based fault injection in multiplications inside a CUDA/C++ quantized convolution routine

number formats, assuming that the gate-level description of a target circuit that uses the desired number representation is available.

B. Optimizations for small bit widths

Quantization to small bit widths offers several opportunities for optimization. For instance, following up on the example presented in Figure 1, a lookup table-based approach can be used to compute quantized convolutions with faulty multiplications much more efficiently. Assuming weights and activations are quantized to INT8, there can only be $256^2 = 65,536$ different multiplications between 8-bit integers.To avoid recomputing the same results over and over, faulty multiplications can be precomputed and stored in a lookup table (LUT) occupying only 64K bytes. In the convolution routine, the `injectable_mul` function can be substituted by a lookup table access, dramatically speeding up the computation compared to repeated calls to the Verilator multiplier model. More in general, LUTs can be used each time quantization is performed with a sufficiently small bit width, and the number of different operations to be performed by the Verilator circuit model is reasonable. The other advantage of the LUT-based approach is that QNN operations like convolutions and matrix multiplications can be parallelized. As a matter of fact, Verilator-generated code is not executable on GPUs, forcing the user to adopt inefficient CPU routines for heavy operations like convolutions. However, a LUT can be easily passed to *ad hoc* GPU routines, effectively enabling *GPU-accelerated fault injections*. An example is given in Figure 2.

IV. Experimental Setup and Results

A. Setup and Fault injector implementation

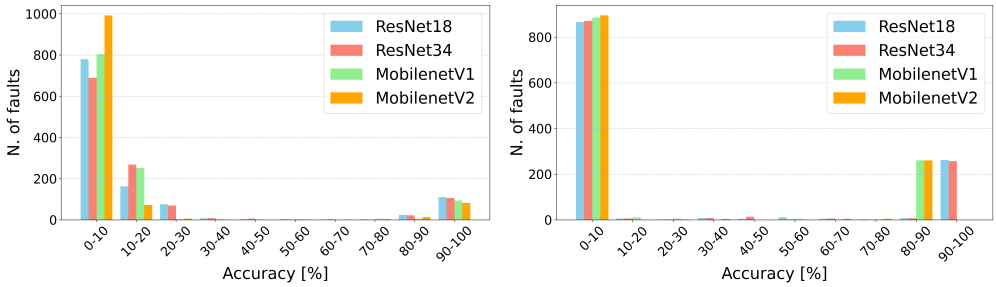
Hardware-level simulation-based FI campaigns have been carried out on a 24-core Intel Core i9-13900K machine, supported by two Nvidia RTX A4000 GPUs, targeting multiplications in the convolutional layers of QNNs executing on X-HEEP [11], a highly customizable RISC-V-based System-on-Chip aimed at ultra-low-power applications. More specifically, the FI campaigns targeted the single-cycle 32-bit integer multiplier inside X-HEEP's CPU, a cve2 (formerly Ibex) RISC-V core, injecting permanent stuck-at faults one at a time in its gate-level description. Tensorflow Lite for Microcontrollers (TFLM), now LiteRT, has been used to perform inferences on the pre-trained and quantized CNN models, and is the target ML framework for fault injections. The QNNs used in the FI campaigns are MobilenetV1, MobilenetV2, ResNet18 and ResNet34 trained on CIFAR-10 and GTSRB datasets and then quantized to INT8 (CIFAR-10 and GTSRB) and UINT8 formats (CIFAR-10 only). The gate-level description of the multiplier has been obtained by synthetizing the design using the 45nm NangateOpenCell library with Synopsys Design Compiler, then Verilator version 5.018 has been used to generate the C++ model of the multiplier. The final injectable multiplier features 1,608 possible fault sites where a stuck-at fault can be injected, for a total of 3,216 possible stuck-at faults. The source code of the TFLM library has been modified to use the fault injection methodology described in Section III by modifying the code relative to convolution routines. The optimizations described in Section III-B have been applied to speed up FI campaigns by leveraging GPU acceleration. For these purposes, a custom CUDA convolution routine has been written. This implementation allows to perform, on our system, a single fault-injected inference in 7-8 ms on average.

B. Evaluation of QNNs

The method described in [12] has been used to obtain a statistically significant fault set. With a 3% error margin and a 99% confidence interval, the minimum number of randomly chosen faults to be injected in order to obtain relevant results is $n = 1174$. The evaluation process focused on two metrics: top-1 and top-5 accuracy. To establish the reference fault-free accuracy values, 2,000 images from the test set of each dataset were chosen as representative samples. The fault-free inferences were performed using the Verilator-based methodology described previously, without injecting any faults.The next step involved performing faulty inferences for each one of the 1,174 multiplier stuck-at faults, using the procedure described in Figure 2: (i) a fault is injected into the DUT; (ii) the lookup table is populated; (iii) the 2,000 samples from the test set are passed to the network and results are collected. To conclude the evaluation, faulty top-1 and top-5 accuracies have been collected for each fault and then averaged over the total number of injected faults. Results are reported in Table I. Data in Table I show how, in case of a permanent stuck-at fault affecting the integer multiplier, there is an impressive degradation in terms of both top-1 and top-5 accuracy, contradicting previous studies on the robustness of QNNs. Based on the reported experiments, on average, the top-1 accuracy degrades by 72.70% in QNNs trained on CIFAR-10. Figure 3 shows the distribution of faulty accuracies for the INT8-quantized networks trained on CIFAR-10 and GTSRB. It is evident how most permanent stuck-at faults tend to either slightly degrade the accuracy, or degrade it to the point of rendering the QNN a random classifier, with almost

TABLE I: Evaluation results of FI campaigns on QNNs trained on CIFAR-10 and GTSRB

Model	Dataset	Quantization Type	Top-1 fault-free accuracy [%]	Avg. top-1 faulty accuracy [%]	Top-5 fault-free accuracy [%]	Avg. top-5 faulty accuracy [%]
MobileNet	CIFAR-10	INT8	91.60	17.89	99.80	56.03
	CIFAR-10	UINT8	91.40	17.85	99.75	55.95
	GTSRB	INT8	88.25	24.01	96.90	38.35
MobileNetV2	CIFAR-10	INT8	90.60	16.92	99.05	54.24
	CIFAR-10	UINT8	90.50	16.88	99.40	54.21
	GTSRB	INT8	89.20	24.37	97.85	37.97
ResNet18	CIFAR-10	INT8	92.80	20.79	99.05	56.58
	CIFAR-10	UINT8	92.80	20.89	99.05	56.75
	GTSRB	INT8	93.10	26.29	97.75	39.45
ResNet34	CIFAR-10	INT8	92.05	20.49	98.30	55.96
	CIFAR-10	UINT8	92.10	20.56	98.45	56.01
	GTSRB	INT8	94.35	26.04	98.85	39.93



(a) CIFAR-10 (b) GTSRB

Fig. 3: Distribution of top-1 accuracies obtained by INT8 QNNs on 2,000 images of the CIFAR-10 and GTSRB test sets. Accuracy values have been obtained by injecting 1,174 permanent stuck-at faults one at a time in the gate-level description of the integer multiplier performing multiplications during convolutions.

nothing in between. In conclusion, while studies performed with application-level or software-level FI campaigns seem to suggest that QNNs are quite robust to faults, a more accurate hardware-level FI campaign shows the opposite situation: QNNs actually seem to be extremely sensible to permanent stuck-at faults affecting a fundamental functional unit.

V. Conclusions

This work described a Verilator-based gate-level fault injection methodology for the reliability assessment of Quantized Neural Networks in presence of permanent stuck-at faults in functional units used during the inference. Furthermore, experiments carried out with the proposed method revealed that the performance of networks quantized to 8-bit signed integers severely degrades in presence of a permanent stuck-at fault in the integer multiplier during convolutions. Future directions include extending the assessment to other layer types and other functional units such as integer ALUs and hardware accelerators for QNNs, as well as extending the Verilator-based approach to floating-point units.

References

[1] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," 2016. [Online]. Available: https://arxiv.org/abs/1602.02830

[2] B. F. Goldstein, S. Srinivasan, D. Das, K. Banerjee, L. Santiago, V. C. Ferreira, A. S. Nery, S. Kundu, and F. M. G. França, "Reliability Evaluation of Compressed Deep Learning Models," in *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*, Feb. 2020, pp. 1–5.

[3] M. A. Neggaz, I. Alouani, S. Niar, and F. Kurdahi, "Are CNNs reliable enough for critical applications? An exploratory study," *IEEE Design & Test*, vol. 37, no. 2, pp. 76–83, Apr. 2020.

[4] F. Libano, P. Rech, B. Neuman, J. Leavitt, M. Wirthlin, and J. Brunhaver, "How Reduced Data Precision and Degree of Parallelism Impact the Reliability of Convolutional Neural Networks on FPGAs," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 865–872, May 2021.

[5] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A White Paper on Neural Network Quantization," Jun. 2021.

[6] (2025) Verilator. [Online]. Available: https://www.veripool.org/verilator/

[7] A. Ruospo, E. Sanchez, L. M. Luza, L. Dilillo, M. Traiola, and A. Bosio, "A Survey on Deep Learning Resilience Assessment Methodologies," *Computer*, vol. 56, no. 2, pp. 57–66, 2023.

[8] G. Gambardella, J. Kappauf, M. Blott, C. Doehring, M. Kumm, P. Zipf, and K. Vissers, "Efficient Error-Tolerant Quantized Neural Network Accelerators," in *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct. 2019, pp. 1–6.

[9] 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," in *2024 25th International Symposium on Quality Electronic Design (ISQED)*, Apr. 2024, pp. 1–8.

[10] M. Beyer, J. M. Borrmann, A. Guntoro, and H. Blume, "Online Quantization Adaptation for Fault-Tolerant Neural Network Inference," in *Computer Safety, Reliability, and Security*. Springer, Cham, 2023, pp. 243–256.

[11] S. Machetti, P. D. Schiavone, T. C. Müller, M. Peón-Quirós, and D. Atienza, "X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller for the Exploration of Ultra-Low-Power Edge Accelerators," 2024.

[12] A. Ruospo *et al.*, "Assessing Convolutional Neural Networks Reliability through Statistical Fault Injections," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.