FORTUNE: A Negative Memory Overhead Hardware-Agnostic Fault TOleRance TechniqUe in DNNs

Samira Nazari[1*], Mahdi Taheri[2,3*], Ali Azarpeyvand[1,2], Mohsen Afsharchi[1],
Tara Ghasempouri[2], Christian Herglotz[3], Masoud Daneshtalab[2,4], and Maksim Jenihhin[2]

[1]University of Zanjan, Zanjan, Iran
[2]Tallinn University of Technology, Tallinn, Estonia
[3]Brandenburgische Technische Universität Cottbus, Cottbus, Germany
[4]Mälardalen University, Västerås, Sweden

***Abstract*—This paper presents FORTUNE, a hardware-agnostic fault tolerance technique for DNNs that leverages quantization to enhance reliability without significant performance overhead. Unlike conventional methods like Triple Modular Redundancy (TMR), which are computationally expensive, the proposed approach uses memory savings from quantization to protect the critical Most Significant Bit, improving fault tolerance in Deep Neural Networks (DNNs). Memory utilization has been reduced by 37.5% across all networks, with vulnerability in AlexNet reduced by 56% compared to the 8-bit version and 84% compared to the unprotected 3-bit version. These improvements come with only a minor increase in execution time of less than 3%. Using AlexNet as an example demonstrates how our approach effectively enhances memory utilization and resilience while causing only a minimal increase in execution time.**

***Index Terms*—deep neural networks, parallel processing, memory overhead, reliability, DNN accelerator**

I. Introduction

As real-time data processing and AI demand grow in embedded systems, Quantized Deep Neural Networks (QNNs) have become vital for deploying models efficiently in resource-constrained environments [1], spanning applications from image classification to safety-critical applications like autonomous driving [2], [3]. QNNs perform complex computations with reduced precision, minimizing computational and memory footprints, and achieving significant energy savings, crucial for energy-constrained platforms [4].

However, quantization introduces challenges in neural network accuracy, especially in critical applications like autonomous systems or medical diagnostics, where precision is non-negotiable [5], [6]. Post-training quantization (PTQ) methods address these challenges, preserving model integrity while reducing bit-widths to enhance efficiency [7]. Yet, QNNs' reliance on extensive memory resources makes them vulnerable to faults, particularly as transistors miniaturize.

979-8-3315-2916-1/24/$31.00 ©2024 IEEE
*Authors contributed equally

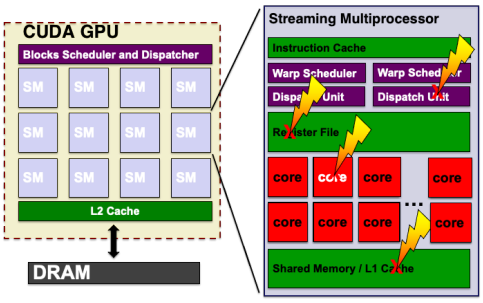

Fig. 1: Fault locations in critical points of a parallel architecture, e.g. the GPU-based [18]

Ensuring fault tolerance is essential, as even minor errors can significantly reduce accuracy [8].

Figure 1 highlights fault locations in critical GPU architecture points. Faults can arise from temperature fluctuations, radiation, aging circuits, and electromagnetic interference [9]. Enhancing fault tolerance in QNNs is thus crucial for safe deployment in safety-critical systems [3], [10], [11].

Traditional fault-tolerant techniques like Triple Modular Redundancy (TMR) introduce significant computational overhead [3]. Selective hardening approaches focus on protecting parameters or neurons with greater impact on the network's output [12]. These methods, however, are mainly applicable to FPGA and ASIC platforms where hardware modifications are possible [13], [14]. For general-purpose CPUs, GPUs, or fixed accelerators, hardware modifications are often infeasible [15].

Other methods, such as Error Correction Codes (ECC) [16], introduce significant memory and computational overhead. Activation restriction techniques mitigate error propagation but are ineffective at high error rates [17].

To address the extensive memory overhead and performance degradation challenges, we present FORTUNE, a model-level hardware agnostic methodology that explores different quantization levels of a DNN model, focusing on reliability, accuracy, memory overhead, and performance aspects. The

proposed methodology introduces a novel fault tolerance technique that uses memory savings from quantization to the Most Significant Bit (MSB) within the same memory elements, maintaining QNN reliability even in the presence of memory faults. A comprehensive GPU-based framework implements this methodology. Specifically, the proposed fault tolerance methodology integrates into the framework's iterative optimization loop, allowing users to define thresholds that balance model accuracy, reliability, and performance. The extensive experiments show that, without this protection, QNNs suffer significant accuracy degradation in fault-prone environments. The results underscore the importance of the proposed technique to minimize memory utilization and optimize protected QNN execution time on DNN accelerators. In this work, without loss of generality, we reported the results for GPU as an example.

The key contributions of this paper are:

- **Negative Overhead Fault Tolerance Technique**: Proposed a protection technique leveraging quantization to triplicate MSB, ensuring robustness against faults.
- **Design Space Exploration Framework**: Developed an open-source framework to assess the impact of quantization on QNN reliability, accuracy, memory utilization, and executing time for design space exploration. https://github.com/nilay1400/DNN-Quantization
- **Introduction of P_{drop} and Reliability-Aware Performance (RAP) metrics**: Introduced P_{drop}, the probability of accuracy drop over a device's lifetime with various BERs, and RAP, a metric for evaluating trade-offs in fault-prone and resource-constrained environments.
- **Validation**: Evaluated the proposed technique and framework on state-of-the-art DNN benchmarks.

The remainder of this paper is structured as follows: Section II presents the proposed methodology. Section III discusses experimental results and the impact on resilience, memory utilization and execution time. Finally, Section IV concludes the paper.

II. Proposed Methodology

Figure 2 illustrates the methodology for quantizing a DNN model, evaluating its reliability in fault-prone environments, and applying the proposed protection. The steps of this methodology are explained by the algorithm 1 and also in detail in this section. The goal is to achieve a quantized model that not only meets predefined accuracy and reliability thresholds but also provides insights into memory utilization and execution time. The algorithm uses a trained FP32 DNN model as input, along with three key parameters: an accuracy threshold a, a reliability threshold b, and a quantization range $[m, n]$ (Algorithm 1 - line 4-6). The accuracy threshold a represents the minimum acceptable accuracy of the model after quantization in the fault-free model, while the reliability threshold b specifies the maximum permissible drop in accuracy due to potential faults. The quantization range $[m, n]$ defines the range of bit widths to be explored during the quantization process.

Algorithm 1 FORTUNE Algorithm

1: **Input:** Trained FP32 DNN model, accuracy threshold a, reliability threshold b, quantization range $[m, n]$
2: **Output:** High-performance Reliable QNN
3: Load the trained DNN
4: SET accuracy_threshold TO a
5: SET reliability_threshold TO b
6: SET quantization range TO $[m, n]$
7: $bit\_width = \frac{n+m}{2}$
8: process = True
9: **while** process **do**
10: quantized_weights← QuantizeWeights(model.weights, bit_width)
11: golden_accuracy←EvaluateAccuracy(model, quantized_weights)
12: **if** $accuracy > a$ AND $accuracy\_drop < b$ **then**
13: Inject faults to weights of the model with different BERs
14: accuracy ← EvaluateAccuracy(model, quantized_weights)
15: accuracy_drop ← golden_accuracy − accuracy
16: Report Memory Utilization
17: Report Execution Time
18: $bit\_width \leftarrow bit\_width - \frac{n-bit\_width}{2}$
19: **else**
20: $bit\_width \leftarrow bit\_width + \frac{n-bit\_width}{2}$
21: **end if**
22: **if** $bit\_width > n$ **then**
23: process = False
24: **end if**
25: **end while**

Step 1: Quantization. The algorithm iterates over each bit width within the specified quantization range. To perform a binary search, the initial bit width considered is the midpoint of the quantization range. Based on the results obtained at this midpoint, the next bit width for weights is selected from either the upper or lower half of the range, depending on whether the desired accuracy and reliability criteria are met (Algorithm 1 - lines 7, 18 and 20). This process is repeated iteratively, narrowing the search range until the optimal bit width is identified. For each selected bit width, the model weights are quantized through linear quantization (Algorithm 1 - line 10). Linear quantization is a widely used technique in model compression, particularly in the context of QNNs. It reduces the precision of weights by mapping a large range of values (typically 32-bit floating-point) to a smaller, fixed range represented by fewer bits. This process involves two main steps: scaling and rounding. The first step in linear quantization is to define the range $[x_{min}, x_{max}]$ within which all the values of the tensor will be mapped.

$$x_{min} = min(x) \tag{1}$$

$$x_{max} = max(x) \tag{2}$$

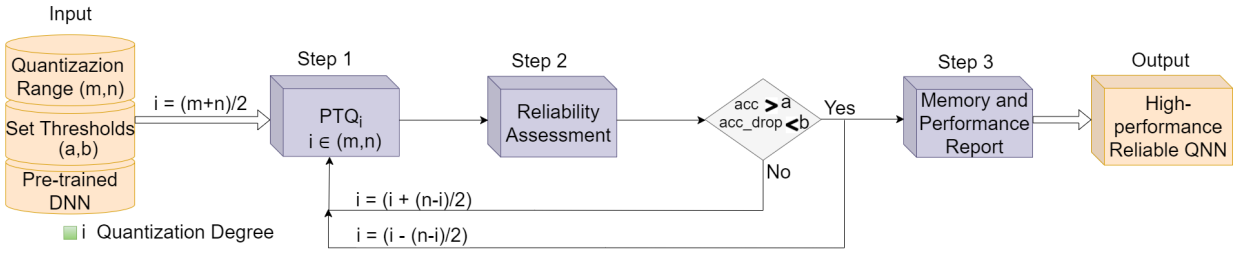

Fig. 2: Proposed methodology for quantizing a DNN model, evaluating its reliability and applying the protection.

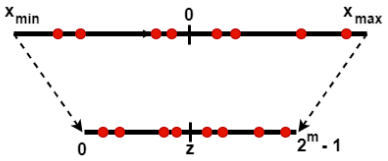

Fig. 3: Affine linear quantization

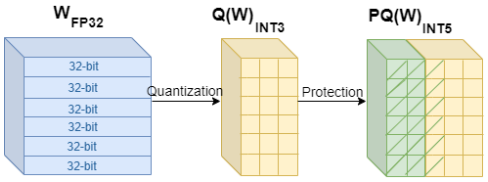

Fig. 4: An example of protected 3-bit weights

Next, the scaling factor s is calculated. This factor determines how the original floating-point values are scaled down to fit within the quantized range. For a given bit width m, the quantized range is $[0, 2^m - 1]$. The scaling factor s is computed as:

$$s = \frac{x_{max} - x_{min}}{2^m - 1} \tag{3}$$

As Figure 3 indicates, the tensor values are then quantized by mapping each value x_i to an integer value q_i within the range $[0, 2^m - 1]$ using the scaling factor s and a zero-point z:

$$z = -\frac{x_{min}}{s} \tag{4}$$

$$q_i = round(\frac{x_i}{s} + z) \tag{5}$$

Here, $round(.)$ denotes rounding to the nearest integer. The result is an integer value q_i that lies within the quantized range. Therefore, weights of the model are converted into unsigned m-bit integers through affine linear quantization.

Then, the accuracy of the model with quantized weights, referred to as the "golden accuracy," is evaluated. This accuracy serves as a baseline for further reliability testing (Algorithm 1 - line 11). If the golden accuracy falls below the predefined threshold a, the algorithm skips further evaluation for this bit width and proceeds to the next (Algorithm 1 - lines 19, 20).

Step 2: Reliability Evaluation and Enhancement. Then, the algorithm proceeds to evaluate its reliability under fault conditions. Faults are incrementally injected into the quantized model's weights, simulating different Bit Error Rates (BERs). After each fault injection, the model's accuracy is re-evaluated, and the accuracy drop is calculated as the difference between the golden accuracy and the current accuracy (Algorithm 1 - lines 13-15).

In the proposed approach, only the MSB bit is protected by replicating it in two redundant bits. During inference, these redundant bits, along with the protected bit, undergo a majority voting process to determine the final value of the protected bit.

TABLE I: Benchmark NNs base accuracies (%)

Type	VGG11	ResNet18	AlexNet	Inception
8-bit	92.41	93.06	95.32	93.70
5-bit	92.19	92.81	95.46	93.24
4-bit	92.18	92.66	94.91	92.03
3-bit	89.83	90.84	93.26	80.71

This method ensures that even with potential faults, the most critical bit remains reliable.

Figure4 illustrates an example of 3-bit weights with a protected bit. In this example, the FP32 weights are first converted into unsigned 3-bit integer values (shown in orange), and the MSB is replicated into two redundant bits (shown in green). Similarly, in 5-bit weights with one protected bit, two redundant bits are added. More generally, in i-bit quantization with one protection bits, two redundant bits are added, and one comparison is performed during inference.

Step 3: Memory Utilization and Execution Time. In this step, the algorithm computes the memory utilization and execution time associated with the given bit width (Algorithm 1 - lines 16, 17). These values provide insights into the trade-offs between quantization, reliability, and resource utilization. The algorithm repeats the above steps for all bit widths within the specified range, ultimately outputting a set of reliable and quantized networks. For each network, detailed reports on execution time and memory utilization are generated, allowing for an informed selection of the optimal quantization configuration.

P_{drop} and RAP

Accuracy drop is evaluated independently of any physical effects that faults might have on memory. To account for these effects on accuracy drop, we define P_{drop} as the probability of experiencing the accuracy drop during the device's lifetime:

$$P_{drop} = N^2 \times W^2 \times T/t \times P_{single} \times BER \times acc\_drop \tag{6}$$

where N is the number of parameters, W is their bit width, T is device life time, t is test time interval, P_{single} is the

TABLE II: Memory Utilization, Execution Time, Vulnerability (accuracy drop due to fault injection) and P_{drop} in DNNs.

Model	Type	Memory Utilization	Executuin Time (%)	Vulnerability (%) {BER}				P_{drop} {BER}			
				1.00E-5	3.00E-5	1.00E-4	3.00E-4	1.00E-5	3.00E-5	1.00E-4	3.00E-4
VGG11	8-bit	225,064,448	100	5.62	22.30	62.40	74.52	50.21E-3	150.63E-3	502.10E-3	1506.30E-3
	P-5-bit	196,931,392	141.72	1.11	5.23	28.33	68.73	35.45E-3	106.35E-3	354.50E-3	1063.52E-3
	P-4-bit	168,798,336	141.72	1.45	6.01	28.19	67.99	25.76E-3	77.30E-3	257.66E-3	773.00E-3
	P-3-bit	140,665,280	141.72	1.49	5.35	34.20	67.13	17.66E-3	53.00E-3	176.68E-3	530.05E-3
ResNet18	8-bit	66,991,616	100	0.18	1.85	14.64	32.29	1.92E-3	5.78E-3	19.27E-3	57.82E-3
	P-5-bit	58,617,664	119.54	0.16	0.81	3.54	18.77	0.85E-3	2.57E-3	8.58E-3	25.74E-3
	P-4-bit	50,243,712	119.54	0.24	0.66	2.63	18.34	0.61E-3	1.84E-3	6.15E-3	18.47E-3
	P-3-bit	41,869,760	119.54	0.25	0.79	3.74	21.89	0.51E-3	1.53E-3	5.10E-3	15.31E-3
AlexNet	8-bit	466,316,032	100	0.00	0.49	3.14	30.71	88.82E-3	266.47E-3	888.23E-3	2664.70E-3
	P-5-bit	408,026,528	102.94	0.14	0.04	0.48	4.73	10.47E-3	31.43E-3	104.78E-3	314.46E-3
	P-4-bit	349,737,024	102.94	0.19	0.10	0.52	5.30	8.62E-3	25.87E-3	86.24E-3	258.72E-3
	P-3-bit	291,447,520	102.94	0.27	0.58	2.62	13.58	15.34E-3	46.03E-3	153.44E-3	460.33E-3
Inception	8-bit	173,011,456	100	0.09	0.16	0.45	1.43	0.57E-3	1.71E-3	5.71E-3	17.15E-3
	P-5-bit	151,234,496	291.73	0.00	0.02	0.02	0.19	0.06E-3	0.18E-3	0.60E-3	1.80E-3
	P-4-bit	129,758,592	291.73	0.01	0.05	0.08	0.14	0.03E-3	0.09E-3	0.33E-3	0.99E-3

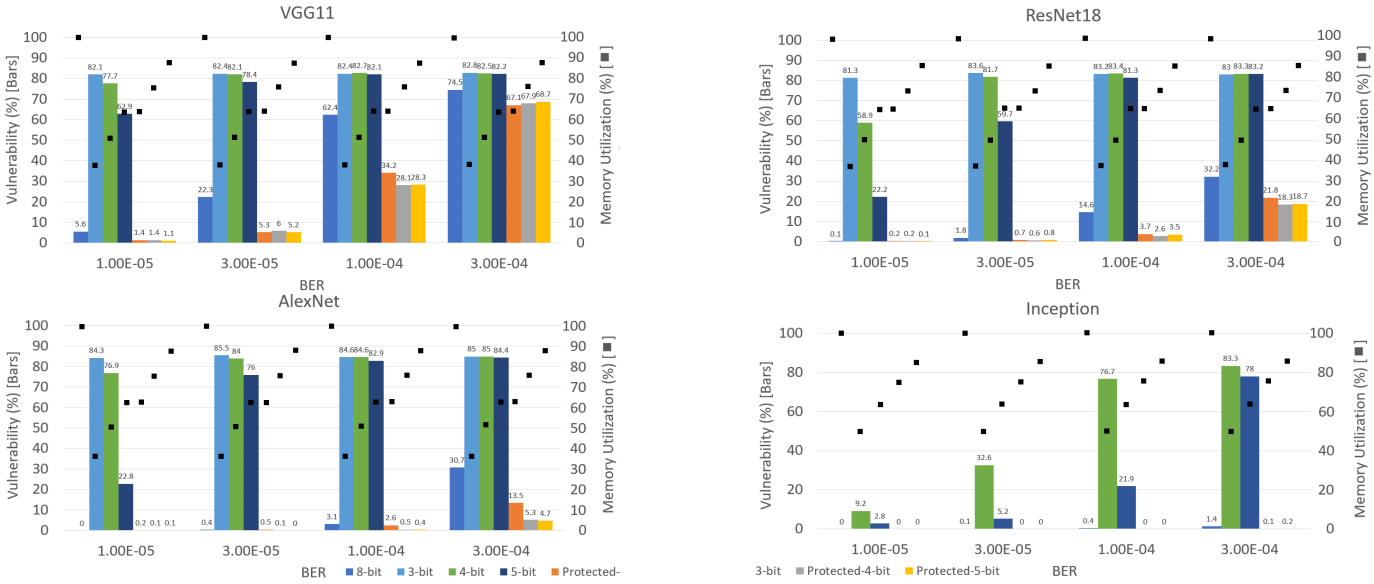

Fig. 5: Vulnerability (accuracy drop due to fault injection) and memory utilization trade-offs in different benchmarks: VGG-11 (CIFAR), ResNet-18 (CIFAR), AlexNet (FashionMNIST) and Inception(CIFAR).

probability of one bit flip during t and acc_drop is the reported accuracy drop in the BER. This metric is based on the probability of a one-bit flip stated in [19]. As the definition suggests, the more resilient the networks are, the smaller the value of P_{drop} becomes.

To account for performance along with accuracy drop and memory footprint, we define the Reliability-Aware Performance (RAP) metric as:

$$RAP = acc\_drop \times mem\_ovh \times perf\_ovh \qquad (7)$$

where acc_drop represents the accuracy drop, $perf\_ovh$ refers to the execution time overhead, and mem_ovh denotes the memory utilization overhead. Smaller RAP values indicate more reliable networks with lower memory and performance overhead.

III. Experimental Results

A. Experimental Setup

To evaluate the reliability and performance of the proposed method, we study four different neural networks. AlexNet is trained on the Fashion MNIST dataset, while VGG11, ResNet-18, and Inception are trained on the CIFAR-10 dataset. Quantization and reliability evaluations are conducted for all networks, resulting in the introduction of different reliable versions for each model. Then, the memory utilization and execution time associated with FORTUNE is assessed. To further quantify the effectiveness of the results, P_{drop} and RAP values are reported for each network to highlight the trade-offs involved.

B. Quantization and reliability evaluation

To facilitate comparison across different networks, the results for four different bit widths are presented in this section.

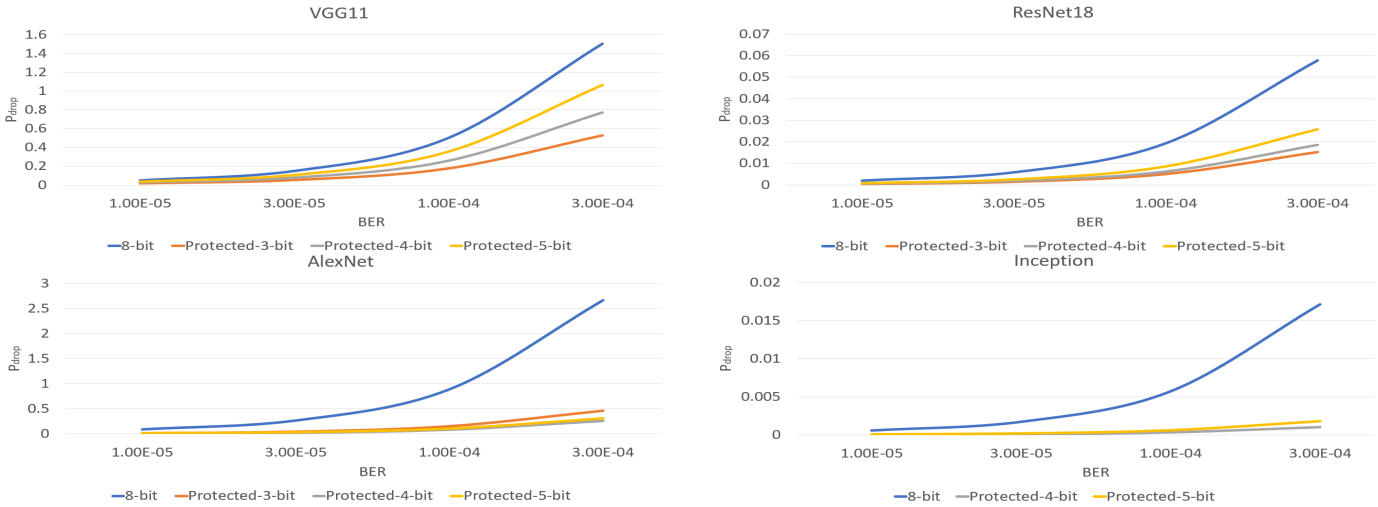

Fig. 6: P_{drop} in VGG-11 (CIFAR), ResNet-18 (CIFAR), AlexNet (FashionMNIST) and Inception(CIFAR).

The unprotected 8-bit version is used as the baseline network. As previously mentioned, the weights are quantized using affine linear quantization, and no retraining is performed. The accuracy of the different quantized networks is presented in Table I. Notably, the accuracy of Inception in its 3-bit version is 80.71%, representing an accuracy reduction of over 10%.

To evaluate reliability, a random fault injection is conducted across all weights in the DNNs under study. The number of injected faults is determined using a BER ranging from 10^{-5} to 3×10^{-4}, covering a comprehensive range of potential errors [8]. Fault injection is repeated several times to reach an acceptable confidence level, following the approach in [20]. This reference provides an equation to reach a 95% confidence level and 1% error margin. For each bit width and BER, the resulting drop in accuracy (with respect to the corresponding fault-free model) is reported in Table II as Vulnerability of the networks. Although the protected versions of AlexNet and ResNet-18 do not exhibit significant differences in Vulnerability values at lower BERs, a considerable difference becomes apparent at higher BERs. Conversely, VGG11 and Inception experience less vulnerability across all protected versions and BER levels. Additionally, the unprotected versions of the quantized networks show worst vulnerabilities, as illustrated in Figure 5.

C. Reliability, memory and performance trade-off

Memory utilization is defined as the combination of the number of parameters and their respective bit widths, while execution time refers to the execution time of each network under study. Memory utilizationn and execution time are reported in Table II, where execution time is normalized with respect to the unprotected 8-bit model. Figure 5 illustrates the trade-off between vulnerability and memory utilization for all protected and unprotected versions of the quantized networks. As the BER increases, all protected models exhibit lower vulnerability compared to the unprotected 8-bit model, while also utilizing less memory. For example, while the unprotected 5-bit model has the same memory utilization as the protected 3-bit model, the protected model demonstrates significantly lower vulnerability. To clarify, all quantized models (5-bit, 4-bit, and 3-bit) are significantly more vulnerable than their protected counterparts. Moreover, the protected versions still maintain lower memory utilization compared to the base 8-bit model.

P_{drop} values for the DNNs under study are reported in Table II and Figure 6. All final networks are considered more resilient when evaluated based on P_{drop}. This indicates that, when factoring in both memory footprint and vulnerability, all protected quantized networks exhibit greater resilience throughout their lifetime.

RAP values are reported in Figure 7. Except for VGG11 at the highest BER, all other networks demonstrate smaller RAP values in protected quantized versions, indicating better trade-off between reliability, memory utilization, and execution time. Smaller RAP values indicate that the networks introduced by FORTUNE are more resilient, while also requiring less memory and execution time compared to the base 8-bit models.

IV. Conclusion

This paper introduces FORTUNE, a novel framework designed to enhance fault tolerance in DNNs through quantization, offering a balanced trade-off between reliability, memory usage, and execution time. By leveraging memory savings to protect the critical Most Significant Bit, FORTUNE improves fault tolerance without the high computational costs of conventional methods like TMR. As examples, we demonstrated memory reductions of 37.5% across networks, with vulnerability in AlexNet reduced by 56% compared to the 8-bit model and 84% compared to the unprotected 3-bit model. These improvements were achieved with less than a 3% increase in execution time. FORTUNE proposes a flexible framework that allows for the identification of the most suitable network configuration, optimizing the trade-off between reliability, memory efficiency, and execution time based on specific application requirements.

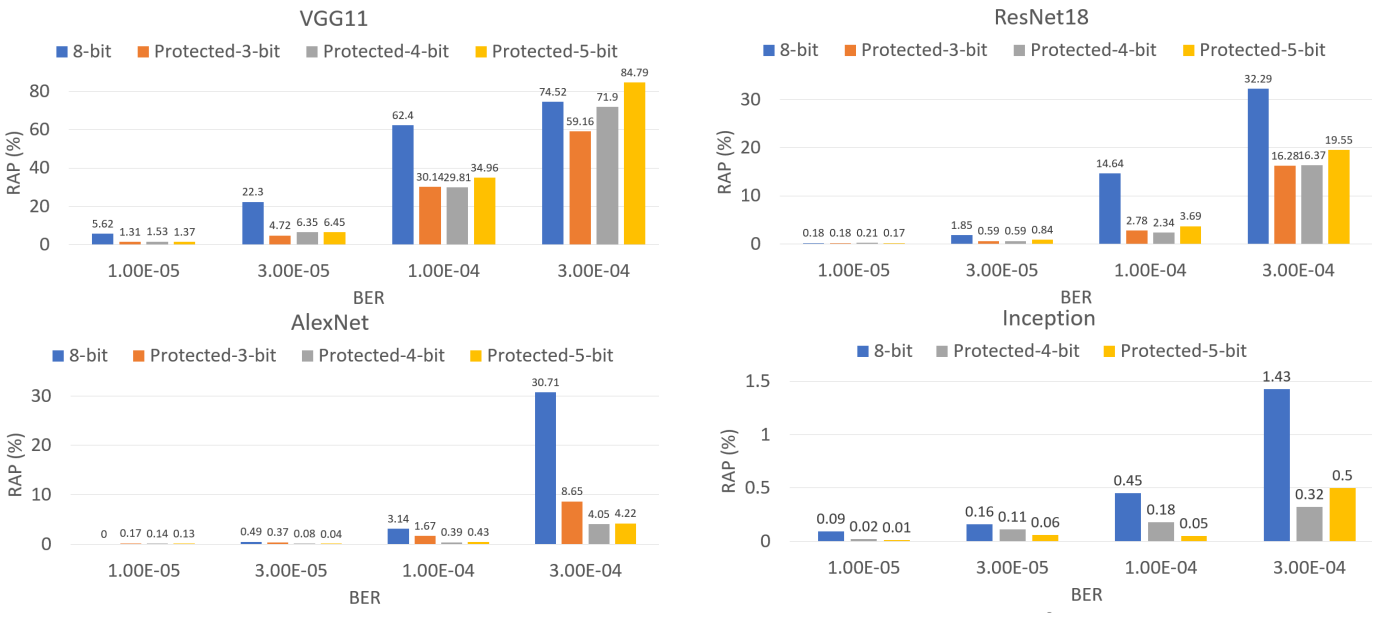

Fig. 7: RAP in VGG-11 (CIFAR), ResNet-18 (CIFAR), AlexNet (FashionMNIST) and Inception(CIFAR) .

V. Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS", EU Grant Project 101160182 "TAICHIP" and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID "458578717".

References

[1] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks in image classification," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 6, pp. 1–50, 2023.

[2] M. Taheri, "Dnn hardware reliability assessment and enhancement," *27th IEEE European Test Symposium (ETS).*, May 2022.

[3] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[4] M. Taheri, M. Riazati, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, J. Raik, M. Sjödin, and B. Lisper, "Deepaxe: A framework for exploration of approximation and reliability trade-offs in dnn accelerators," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2023, pp. 1–8.

[5] 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)*. IEEE, 2024, pp. 1–8.

[6] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. D. Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, E. Sanchez, and M. Taheri, "Special session: Approximation and fault resiliency of dnn accelerators," in *2023 IEEE 41st VLSI Test Symposium (VTS)*, 2023, pp. 1–10.

[7] Z. Yuan, J. Liu, J. Wu, D. Yang, Q. Wu, G. Sun, W. Liu, X. Wang, and B. Wu, "Benchmarking the reliability of post-training quantization: a particular focus on worst-case performance," *arXiv preprint arXiv:2303.13003*, 2023.

[8] M. H. Ahmadilivani *et al.*, "Enhancing fault resilience of qnns by selective neuron splitting," in *2023 IEEE 5th AICAS*, 2023, pp. 1–5.

[9] M. Taheri, M. H. Ahmadilivani, M. Jenihhin, M. Daneshtalab, and J. Raik, "Appraiser: Dnn fault resilience analysis employing approximation errors," in *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2023, pp. 124–127.

[10] M. H. Ahmadilivani *et al.*, "Special session: Reliability assessment recipes for dnn accelerators," in *2024 VTS*. IEEE, 2024, pp. 1–6.

[11] M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, and A. Bosio, "Saffira: a framework for assessing the reliability of systolic-array-based dnn accelerators," in *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2024, pp. 19–24.

[12] A. Ruospo and E. Sanchez, "On the reliability assessment of artificial neural networks running on ai-oriented mpsocs," *Applied Sciences*, vol. 11, no. 14, p. 6455, 2021.

[13] M. Taheri, N. Cherezova, S. Nazari, A. Rafiq, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "Adam: Adaptive fault-tolerant approximate multiplier for edge dnn accelerators," in *2024 IEEE European Test Symposium (ETS)*, 2024, pp. 1–4.

[14] M. Taheri, N. Cherezova, S. Nazari, A. Azarpeyvand, T. Ghasempouri, M. Daneshtalab, J. Raik, and M. Jenihhin, "Adam: Adaptive approximate multiplier for fault tolerance in dnn accelerators," *Authorea Preprints*, 2024.

[15] M. Nourazar, V. Rashtchi, A. Azarpeyvand, and F. Merrikh-Bayat, "Code acceleration using memristor-based approximate matrix multiplier: Application to convolutional neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2684–2695, 2018.

[16] S. Lee and J. Yang, "Value-aware parity insertion ecc for fault-tolerant deep neural network," in *2022 DATE*, 2022, pp. 724–729.

[17] B. Ghavami *et al.*, "Fitact: Error resilient deep neural networks via fine-grained post-trainable activation functions," in *2022 DATE*, 2022, pp. 1239–1244.

[18] R. G. Alía, A. Coronetti, K. Bilko, M. Cecchetto, G. Datzmann, S. Fiore, and S. Girard, "Heavy ion energy deposition and see intercomparison within the radnext irradiation facility network," *IEEE Transactions on Nuclear Science*, vol. 70, no. 8, pp. 1596–1605, 2023.

[19] Z. Yan, Y. Shi, W. Liao, M. Hashimoto, X. Zhou, and C. Zhuo, "When single event upset meets deep neural networks: Observations, explorations, and remedies," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 163–168.

[20] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 502–506.