MONO: Enhancing Bit-Flip Resilience With Bit Homogeneity for Neural Networks

Maryam Eslami, Yuhao Liu, *Graduate Student Member, IEEE*, Salim Ullah, Mostafa E. Salehi, Reshad Hosseini, Seyed Ahmad Mirsalari, and Akash Kumar, *Senior Member, IEEE*

***Abstract*—*Deep neural networks* (DNNs) have been applied across diverse domains, including safety-critical applications. Past studies indicate that DNNs are very sensitive to changes in weights and activations due to uneven bit-weight distribution in standard number formats like fixed points, which can cause significant output accuracy fluctuations. To address this issue, we introduce a new data type called MONO to enhance bit-flip resilience using uniformity at the bit level by employing symmetric weights for all bit positions. On average, MONO has improved error resilience more effectively than the fixed-point data type, even when utilizing *triple modular redundancy* (TMR) and most significant bit (MSB) protection, while maintaining low overhead.**

***Index Terms*—Deep neural network (DNN), error resilience, fault Injection, hardware fault, homogeneity.**

I. Introduction

DEEP neural networks (DNNs) are widely applied in real-life domains, such as autonomous driving and industrial controls, where safety is paramount. DNN face two major challenges when used in resource-constrained embedded systems for safety-critical applications: 1) high resource consumption and memory access requirements and 2) vulnerability to hardware errors, which can have severe consequences.

To tackle the first issue, various quantization schemes have been developed to reduce power and memory consumption by lowering the bit precision of network parameters, converting floating points to 32-bit or as low as 1-bit fixed-point systems, leading to *binary neural networks* (BNNs). Additionally, Ghasemzadeh et al. [1] introduced the *multiple-level binary* (MLB) approach within the *RebNet* framework to improve memory efficiency further while preserving the model accuracy.

Manuscript received 12 August 2024; accepted 12 August 2024. This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG) through the Project LeanMICS under Grant 534919862. This manuscript was recommended for publication by A. Shrivastava. *(Corresponding author: Maryam Eslami.)*

Maryam Eslami is with the Chair of Processor Design, CFAED, TU Dresden, 01062 Dresden, Germany (e-mail: maryam.eslami@tu-dresden.de).

Yuhao Liu is with the Chair of Processor Design, CFAED, TU Dresden, 01062 Dresden, Germany, and also with the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), 04105 Leipzig, Germany (e-mail: yuhao.liu1@tu-dresden.de).

Salim Ullah is with the Chair of Embedded Systems, Ruhr-Universität Bochum, 44801 Bochum, Germany (e-mail: salim.ullah@rub.de).

Mostafa E. Salehi, Reshad Hosseini, and Seyed Ahmad Mirsalari are with the School of Electrical and Computer Engineering, University of Tehran, Tehran 14395-515, Iran (e-mail: mersali@ut.ac.ir; reshad.hosseini@ut.ac.ir; ahmad.mirsalari@ut.ac.ir).

Akash Kumar is with the Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI), 04105 Leipzig, Germany, and also with the Chair of Embedded Systems, Ruhr-Universität Bochum, 44801 Bochum, Germany (e-mail: akash.kumar@rub.de).

Digital Object Identifier 10.1109/LES.2024.3444921

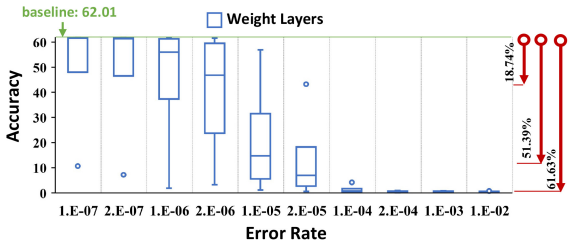

Fig. 1. Classification accuracy of *ResNet-18* trained on TinyImageNet under different error rates.

To address the second concern, the fixed-point data type incorporates static bit-weights across diverse bit positions (i.e., 2^{n-1}, 2^{n-2}, ..., 0), and *RebNet* uses dynamic trainable bit weights (i.e., γ_1, γ_2, ..., γ_m) to represent the DNNs parameters. Previous research has shown that DNNs using fixed-point data types can experience significant accuracy loss due to bit errors in asymmetric bit-weights, which can cause numerical perturbations that greatly exceed the original values [2], [3], [4]. Moreover, using asymmetric dynamic bit-weights also poses reliability challenges for the *RebNet* scheme.

According to Fig. 1, the accuracy of *ResNet-18* utilizing fixed-point data type, can drop from 62.01% to 10.62% at the lowest error rate, i.e., 10^{-7}. The insights from Fig. 1 motivate us to equalize the bit weights of parameters by removing both the *most significant bit* (MSB) and the *least significant bit* (LSB) across the neural network. This adjustment aims to enhance the network's capacity for error resilience.

Contributions: This letter proposes an innovative scheme with uniform bit-weights for network parameters based on quantization-aware training inspired by stochastic computing. This novel data type enhances reliability and attains uniform bit positions in weight representation. To our understanding, no prior research has explored the uniform distribution of resilience across a DNN by equating parameter bit-weights to enhance error robustness while upholding accuracy levels.

1) This letter introduces a highly accurate data type called MONO based on the homogeneity of the bit-weights of the parameter representation to improve the error resilience of neural networks by introducing redundancy.
2) We developed a framework for training DNNs based on the MONO data type and injecting faults in the weight/activation parameters to assess error resilience.
3) We explore the hardware implementation of MONO-based processing elements for neural network accelerators on FPGA and evaluate the critical path delay (CPD), resource consumption, and dynamic power.

II. Related Work

Earlier studies investigated the role of redundancy in improving the error resilience of DNNs, such as *triple module*

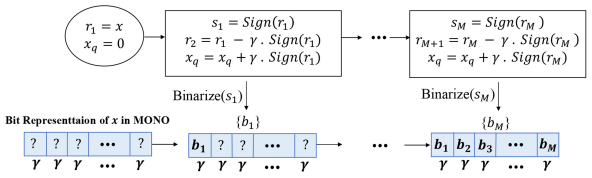

Fig. 2. Schematic flow for computing the M-level bit representation of the input x in the MONO data type.

redundancy (TMR) to mitigate the effect of errors at the expense of 3× area [2]. In addition, several previous studies have examined the effect of different common data types on the reliability of DNNs [5], [6], [7]. These studies show that DNNs have some fault resilience, but it varies with the bit position weights in network parameters. One study found that using a wider range of integer values in fixed-point data types increases failure risk [5]. Ruospo et al. [6] examined CNN reliability using a range of bit weights in floating-point and fixed-point data types. They conclude that asymmetric bit weight representation significantly impacts the CNN model reliability.

In a separate research track, error resilience of DNNs can vary across different parts of the neural network, such as neurons, layers, and parameters of models [5], [8]. Schorn et al. [9] proposed an analytical method for predicting the error resilience of neurons in DNNs to assign more critical neurons to protected hardware elements. This mapping technique decreases the flexibility of DNN accelerators and adds additional overhead. Schorn et al. [10] introduced a feature resilience optimization technique to equalize the intralayer error resilience within DNNs by adjusting the DNNs weights to prevent the need for tolerating additional overheads.

III. MONO: Bit Homogeneity for Neural Networks

Proposed Approach (MONO Data Type): MONO is built based on the multilevel binary scheme. The bit representation of a fixed-point input x in the MONO data type is depicted in Fig. 2. This scheme approximates the input x with multilevel binary value x_q based on Fig. 2. For the presenting M-level binary value in the MONO data type, we need M scaling factors $\{\gamma_1, \gamma_2, \ldots, \gamma_M\}$. These scaling factors are always positive and equal to each other, i.e., $\gamma_1 = \gamma_2 = \cdots = \gamma_M = \gamma$ in the MONO scheme.

Note that, we have one γ_w for all weights and one γ_a for all activations corresponding to a specific layer.

Batch-Normalization Layer in the MONO Data Type: Usually, a batch-normalization layer normalizes the output of the dot product $u = \text{dot}(\vec{a}, \vec{w})$ before passing it to an activation function, like ReLU. This layer converts u into $\alpha_{\text{BN}} \times u - \beta_{\text{BN}}$, where α_{BN} and β_{BN} are the trainable parameters of the layer. We can rewrite the dot-product u in the MONO scheme based on (1). After applying batch-normalization, u is converted into $\alpha_{\text{BN}}\gamma_{\text{in}}u' + \beta_{\text{BN}}$ in the MONO scheme. This simplification helps us to improve hardware implementation clarified in Section IV-B

$$u = \text{dot}(\vec{a}, \vec{w}) = \text{dot}\left(\sum_{i=1}^{M} \gamma_w \vec{s}_{w_i}, \sum_{j=1}^{M} \gamma_a \vec{s}_{a_j}\right)$$

$$= \gamma_w\gamma_a \text{dot}\left(\sum_{i=1}^{M} \vec{s}_{w_i}, \sum_{j=1}^{M} \vec{s}_{a_j}\right) = \gamma_w\gamma_a u' = \gamma_{\text{in}} u'. \quad (1)$$

Activation Function and Encoder in the MONO Data Type: The ReLU [11] function is commonly used in standard DNNs, such as *ResNet* [12] and *MobileNet* [13] to apply nonlinearity for calculations. Therefore, ReLU serves as the activation function in the processing element of the MONO data type. However, our framework allows us to use any activation function based on the input neural network architecture.

The encoder based on the MONO scheme uses the sign function and a trainable scaling factor to convert a fixed-point input to a multilevel binary value. For each layer, a dedicated scaling factor γ is employed to quantize the activations. These scaling factors must be learned during the training phase.

Multilevel Popcount in MONO: In MONO, we can calculate the dot product of an M-level residual binary weight vector $\vec{w}$ and an M-level residual binary activation vector $\vec{a}$ using Popcount operation. Let weight vector and activation vector be $\vec{w} = \sum_{i=1}^{M} \gamma_w \vec{s}_{w_i}$ and $\vec{a} = \sum_{i=1}^{M} \gamma_a \vec{s}_{a_i}$, respectively. $\vec{s}_{w_i}$ and $\vec{s}_{a_i}$ denote the ith residual sign vector of the weight and activation, respectively. Dot product operation between M-level binarized weight and M-level binarized activation can be performed according to (2). In (2), $\vec{b}_{w_i}, \vec{b}_{a_j}$ are encoded bit corresponding to $\vec{s}_{w_i}, \vec{s}_{a_j}$, respectively

$$u = \text{dot}(\vec{a}, \vec{w}) = \text{dot}\left(\sum_{i=1}^{M} \gamma_w \vec{s}_{w_i}, \sum_{j=1}^{M} \gamma_a \vec{s}_{a_j}\right)$$

$$= \gamma_w\gamma_a \text{dot}\left(\sum_{i=1}^{M} \vec{s}_{w_i}, \sum_{j=1}^{M} \vec{s}_{a_j}\right)$$

$$= \gamma_w\gamma_a \text{dot}\left(\sum_{i=1}^{M} \vec{b}_{w_i}, \sum_{j=1}^{M} \vec{b}_{a_j}\right) = \gamma_{\text{in}} \text{dot}\big(w_{\text{pop}}, a_{\text{pop}}\big). \quad (2)$$

A. Training DNN Models in MONO

This section uses the MONO scheme to calculate gradients for training DNN parameters, including weights, activations, and scaling factors. We denote $\mathcal{L}$ as a cost function. In addition, x_w and x_a are full precision weight and activation, respectively, which are approximated as $x_{w-q} = \sum_{i=1}^{M} \gamma_w \text{Sign}(r_{w_i})$ and $x_{a-q} = \sum_{j=1}^{M} \gamma_a \text{Sign}(r_{a_j})$ using the MONO data type. In this equation, r_{w_i} and r_{a_j} denote the ith and jth residual error for weight and activation, respectively. Equations 3 and 4 describe how to compute the derivative of the cost function with respect to γ_w and the weight

$$\partial\mathcal{L}\gamma_w = \partial\mathcal{L}x_{w-q}\partial x_{w-q}\gamma_w = \partial\mathcal{L}x_{w-q} \sum_{i=1}^{M} \text{Sign}\big(r_{w_i}\big) \quad (3)$$

$$\partial\mathcal{L}x_w = \partial\mathcal{L}x_{w-q}\partial x_{w-q} \sum_{i=1}^{M} \text{Sign}\big(r_{w_i}\big) \sum_{i=1}^{M} \partial\text{Sign}\big(r_{w_i}\big) r_{w_i}$$

$$= \partial\mathcal{L}x_{q-w} \times M \times \gamma_w \times 1_{|r_{w_i}|\leq 1} \quad (4)$$

where the derivative term $\partial\text{Sign}(x)x$ is 1 if $|x| \leq 1$, else it is 0. These equations can be extended to include activation and γ_a.

IV. Experimental Analysis

A. Experimental Analysis

We demonstrated the improvements in error resilience by our scheme with three DNN models and two datasets. We explore varied bit-widths in neural networks using MONO, selecting optimal bit-widths based on fixed-point that maintain network accuracy less than 1% compared to the floating-point precision. This leads us to consider 5, 7, and 8-bit for *ResNet-20* [12], *MobileNet-V1* [13], and *ResNet-18* [12] models, respectively. For training MONO and MLB, we use the *Adam* optimizer with an initial learning rate of

TABLE I
TOLERATED BER FOR VARIOUS DATA TYPES AND RESILIENCE METHODS IN THREE DNN MODELS ON DIFFERENT BENCHMARKS

Network	Tolerated Weight BER					Tolerated Activation BER				
	MONO	MLB	FxP	TMR	MSB-protection	MONO	MLB	FxP	TMR	MSB-protection
ResNet-20 [2] (*CIFAR-10* [7])	10^{-4}	10^{-5}	10^{-5}	10^{-4}	4×10^{-5}	2×10^{-4}	4×10^{-5}	2×10^{-6}	10^{-5}	4×10^{-6}
MobileNet-V1 [3] (*CIFAR-10* [7])	2×10^{-5}	2×10^{-6}	2×10^{-6}	10^{-5}	10^{-5}	2×10^{-3}	10^{-3}	2×10^{-5}	10^{-4}	10^{-4}
ResNet-18 [2] (*Tiny ImageNet* [4])	2×10^{-6}	4×10^{-7}	-	4×10^{-7}	4×10^{-7}	4×10^{-3}	4×10^{-3}	4×10^{-5}	4×10^{-4}	2×10^{-4}

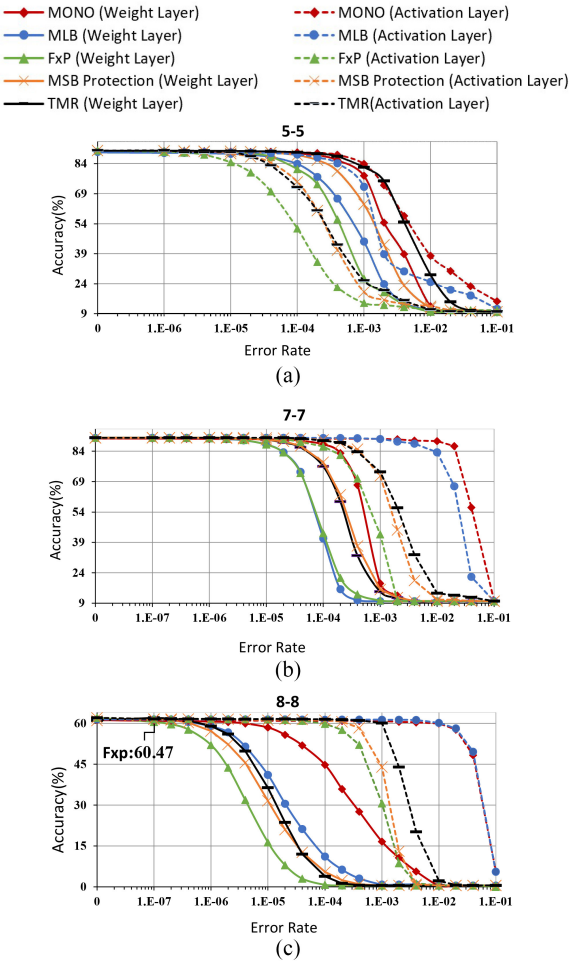

Fig. 3. Comparison of bit-error resilience improvements in three DNN models on different benchmarks. (a) ResNet-20 on CIFAR-10. (b) MobileNet V1 on CIFAR-10. (c) ResNet-18 on TinyImageNet.

0.01/0.01/0.0001, multiplied by 0.1 after (1/3)/(5/6)/(1/6) of 300/300/180 epochs in *ResNet-20*/*MobileNet-V1*/*ResNet-18* models. We also use a warmup scheduler to restart the learning rate after 90 epochs to improve accuracy in *ResNet-18* on *TinyImageNet* [15]. We retrain *ResNet-20*/*MobileNet-V1*/*ResNet-18* models based on fixed-point using the *Adam* optimizer with an initial learning rate of 0.01/0.01/0.0001, reduced tenfold after (1/3) of 60/60/60 epochs.

In our simulation environment, errors can be injected into both DNN weights and activations. Weight errors are injected before inference, while activation errors are performed during inference. Following the specification of the desired error rate, the framework randomly selects the affected bit positions of the erroneous variables (weight/activation) and executes inference to assess accuracy under the applied perturbations. We repeat all the experiments at the desired error rates until statistically consistent results are achieved.

We investigate the error resilience of target models to errors injected into model parameters and assess the classification accuracy of the test set across various bit error rates (BERs). The accuracy curves are illustrated in Fig. 3. Furthermore, we illustrate the tolerated BER across different data types and resilience methods in Table I. Tolerated BER is defined as a point where the accuracy drops more than 1% in the subsequent point. according to Table I, we have different improvements for different models trained on various datasets. For instance, according to Table I, *ResNet-20* and *MobileNet-V1* based on the MONO data type can tolerate up to **10**$\times$ more errors compared to fixed-point and MLB data types if all errors happen on weights in the *CIFAR-10* benchmark. The MONO datatype significantly reduces activation errors and is more effective than the traditional approaches like TMR and MSB protection, which have high overhead. For instance, the MONO approach demonstrates up to (**100/10/20**)$\times$ more activation error resilience with equivalent accuracy loss for *ResNet-18* utilizing fixed-point/TMR/MSB protection, respectively.

We also explore the impact of fault injection with the error rates ($10^{-7} \sim 2\times10^{-4}$) affecting both weights and activation layers across all trials for each error rate, as depicted in the box plot in Fig. 4. According to this figure, MLB and fixed-point data types experience extensive misclassifications, especially in the weight layer. For instance, the accuracy of *ResNet-18* based on the fixed-point/MLB data type can drastically drop **51.4**% (from 62.02% to 10.62%)/**10.18**% (from 61.45% to 51.28%) in the smallest error rate, i.e., 10^{-7} on the weight layer. The use of TMR and MSB protection also results in accuracy drops of **15.41**% (from 62.02% to 46.61%) and **46.68**% (from 62.02% to 15.34%), respectively. In contrast, the MONO datatype shows only a **0.83**% accuracy drop (from 61.18% to 60.35%) under the same conditions. In addition, in the worst-case scenario for the activation layer, the accuracy drop of *ResNet-18* for the MONO datatype is **0.72**%, compared to **8.79**%, **1.22**%, and **2.61**% for fixed-point, TMR, and MSB protection, respectively, at an error rate of 2×10^{-4}.

Calculation of MONO Overhead: Consider an m-bit model with W weights, A activations, and L layers (convolution and fully connected). Our experiments reveal that 8-bit is sufficient to quantize scaling factors, γ_w and γ_a. The model redundancy for MONO, MLB, TMR, and MSB-protection requires $16L$, $16mL$, $2m(W+A)$, and $(W+A)$ extra bits, respectively, compared to the fixed-point model. We disregard the cost of the voter in TMR and duplicate the sign bit to implement the MSB-protection technique. For instance, *ResNet-20* (quantized 5-bit on *CIFAR-10*) based on MONO, MLB, TMR, and MSB-protection consumes **0.006**%, **0.03**%, **200**%, and **20**% extra bit, respectively, compared to the fixed-point. It indicates that MONO has the least redundancy overhead among the others.

B. Implementation of the Processing Element Unit in the MONO Datatype

We synthesized and deployed a processing element based on MONO and another based on fixed-point methodology separately on the *Ultra96 FPGA* platform to investigate and compare their hardware designs. Fig. 5 shows the structure of the above-mentioned two implementations, and Table II shows their hardware resource consumption, CPD, and *dynamic*

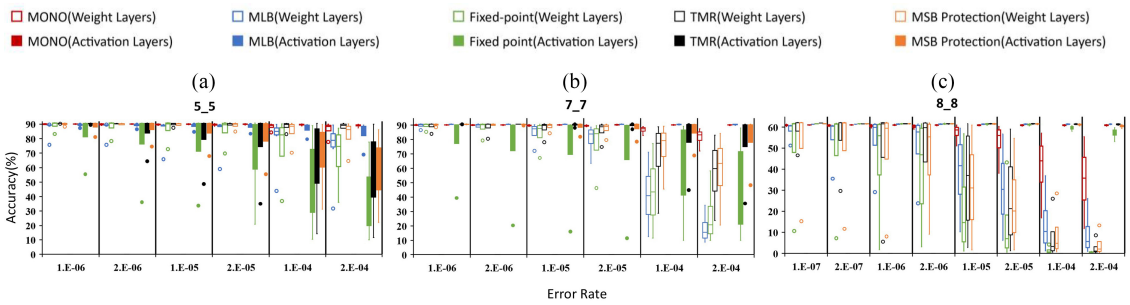

Fig. 4. Impact of fault injection on three different data types (MONO, MLB, and fixed point) and two various protection techniques, on two individual groups of layers 1) weight layer and 2) activation layer, across three different DNNs on two datasets. (a) ResNet-20 on CIFAR-10. (b) MobileNet-V1 on CIFAR-10. (c) ResNet-18 on TinyImageNet.

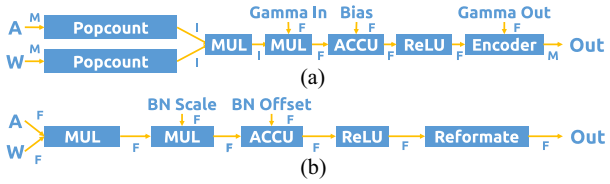

Fig. 5. Hardware structures of MONO-based and fixed-point-based processing element. *A* and *W* are the activation and weight inputs. *M* means MONO format, *I* means signed integer format, and *F* means signed fixed-point format. (a) Hardware structure of MONO-based processing element. (b) Hardware structure of fixed-point-based processing element.

TABLE II
IMPLEMENTATIONS OF MONO-BASED AND FIXED-POINT-BASED PROCESSING ELEMENT ON ULTRA96 FPGA

	Input Width	Frequency	LUT	FF	CPD	DPC
MONO	8 bits	200 MHz	455	203	4.478 ns	0.013 W
FxP	8 bits	200 MHz	617	172	4.722 ns	0.016 W

power consumption (DPC). The MONO processing element consists of two *Popcount* modules, two multipliers, one accumulator, one ReLU activation module, and one encoder. The inputs of this instance are 8-bit MONO-format activation and weight. The *Popcount* module converts two inputs as 5-bit signed integer values. Two multipliers calculate the product of the activation, weight, and the *batch normalization* folded (BN-folded) input gamma within the MONO data type. As referred in Section III, the input gamma, γ_{in}, can be folded with the scaling parameter of BN, α_{BN}. Therefore, in our hardware implementation, the BN-folded input gamma and bias apply the 16-bit fixed-point format with an 8-bit fraction. The output encoder will convert the result of the ReLU activation function to 8-bit MONO format based on one 8-bit output gamma with a 4-bit fraction following the scheme shown in Fig. 2. Compared with MONO-based processing elements, the fixed-point-based processing element applies the same input (8-bit) and BN parameter (16-bit) data width for a fair comparison with our MONO instance. Reformat module will relimit and reformat the result out ReLU activation as 8-bit fixed-point output. The two instances above are implemented as the pipeline architecture for high clock frequency. Table II shows the synthesis and implementation result of our two instances. Compared with the 8bits × 8bits multiplier for activation and weight computing and 16bits × 16bits multiplier for batch normalization in the fixed-point instance, our MONO design only required one 5bits × 5bits multiplier for pop-counted activation and weight computing and 10bits × 16bits multiplier for BN-folded gamma computing. Hence, MONO processing element utilizes fewer hardware resources for an equivalent input bit-width when compared to the fixed-point instance. As shown in Table II, the MONO processing element instance also demonstrates reduced CPD and DPC in comparison to the fixed-point implementation.

V. CONCLUSION

This letter introduces a novel data type called MONO, which uses quantization-aware training to unify bit-weights across positions. Extensive experiments with various DNNs on image recognition benchmarks confirm that MONO significantly enhances error resistance compared to the conventional data types like fixed-point and MLB, as well as the traditional protection methods like TMR and MSB protection. Additionally, a hardware architecture for a processing element based on the MONO data type is proposed, showing lower area, delay, and power consumption compared to the fixed-point data type.

REFERENCES

[1] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "ReBNet: Residual binarized neural network," in *Proc. IEEE Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2018, pp. 57–64.
[2] N. Khoshavi, A. Roohi, C. Broyles, S. Sargolzaei, Y. Bi, and D. Z. Pan, "SHIELDeNN: Online accelerated framework for fault-tolerant deep neural network architectures," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.
[3] E. Ozen and A. Orailoglu, "SNR: Squeezing numerical range defuses bit error vulnerability surface in deep neural networks," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 5, pp. 1–25, 2021.
[4] F. Koc, B. Salami, O. Ergin, O. Unsal, and A. C. Kestelman, "Can we trust undervolting in FPGA-based deep learning designs at harsh conditions?" *IEEE Micro*, vol. 42, no. 3, pp. 57–65, Jun. 2022.
[5] B. Reagen et al., "Ares: A framework for quantifying the resilience of deep neural networks," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2018, pp. 1–6.
[6] A. Ruospo, A. Bosio, A. Ianne, and E. Sanchez, "Evaluating convolutional neural networks reliability depending on their data representation," in *Proc. IEEE DSD*, 2020, pp. 672–679.
[7] 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 *Proc. IEEE Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 163–168.
[8] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. ACM SC*, 2017, pp. 1–12.
[9] C. Schorn, A. Guntoro, and G. Ascheid, "Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators," in *Proc. IEEE Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2018, pp. 979–984.
[10] C. Schorn, A. Guntoro, and G. Ascheid, "An efficient bit-flip resilience optimization method for deep neural networks," in *Proc. IEEE Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2019, pp. 1507–1512.
[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.
[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.
[13] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[14] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.
[15] A. G. Howard et al., "Tinyimagenet." 2017. [Online]. Available: http://cs231n.stanford.edu/tiny-imagenet-200.zip